SPRINT: High-Throughput Robust Distributed Schnorr Signatures

Fabrice Benhamouda*[1], Shai Halevi*[1], Hugo Krawczyk*[1], Yiping Ma[2], and Tal Rabin*[1,2]

[1]AWS, USA
[2]University of Pennsylvania, USA

March 31, 2024

Abstract

We describe robust high-throughput threshold protocols for generating Schnorr signatures in an asynchronous setting with potentially hundreds of parties. The protocols run a single message-independent interactive ephemeral randomness generation procedure (i.e., DKG) followed by *non-interactive* signature generation for multiple messages, at a communication cost similar to one execution of a synchronous non-robust protocol in prior work (e.g., Gennaro et al.) and with a large number of parties (ranging from few tens to hundreds and more). Our protocols extend seamlessly to the dynamic/proactive setting where each run of the protocol uses a new committee with refreshed shares of the secret key; in particular, they support large committees periodically sampled from among the overall population of parties and the required secret state is transferred to the selected parties. The protocols work over a broadcast channel and are robust (provide guaranteed output delivery) even over asynchronous networks.

The combination of these features makes our protocols a good match for implementing a signature service over a public blockchain with many validators, where guaranteed output delivery is an absolute must. In that setting, there is a system-wide public key, where the corresponding secret signature key is distributed among the validators. Clients can submit messages (under suitable controls, e.g., smart contracts), and authorized messages are signed relative to the global public key.

Asymptotically, when running with committees of n parties, our protocols can generate $\Omega(n^2)$ signatures per run, while providing resilience against $\Omega(n)$ corrupted nodes and broadcasting only $O(n^2)$ group elements and scalars (hence $O(1)$ elements per signature).

We prove the security of our protocols via a reduction to the hardness of the discrete logarithm problem in the random oracle model.

*Work done prior to joining Amazon, partially while at the Algorand Foundation.

Contents

1 Introduction **1**
- 1.1 Other Techniques . . . 3
- 1.2 Prior Work . . . 5
- 1.3 Subsequent Work . . . 6
- 1.4 Organization . . . 7

2 Technical Overview **8**
- 2.1 Starting Point: The GJKR Protocol . . . 8
- 2.2 The Agreement Protocol . . . 9
- 2.3 Signing Many Messages in Parallel . . . 10
- 2.4 Using Super-Invertible Matrices . . . 10
- 2.5 Using Packed Secret Sharing . . . 12
- 2.6 More Efficient Signing . . . 12
- 2.7 The Dynamic Setting . . . 13
- 2.8 Sub-sampling the Committees . . . 14
- 2.9 More Optimizations . . . 14
- 2.10 Parameters and Performance . . . 14

3 The SPRINT Protocols **17**
- 3.1 Static-Committee Setting . . . 17
- 3.2 The Dynamic/Proactive Setting . . . 18

4 The Agreement Protocol **18**
- 4.1 Agreement in SPRINT, the Static Case . . . 21
- 4.2 Agreement in the Dynamic/Proactive setting . . . 22

A More Optimizations **26**

B Faster Multiplication by a Super-Invertible Matrix **29**
- B.1 First Solution: Small-Scalar Vandermonde . . . 29
- B.2 Second Solution: ECFFT-EXTEND . . . 29
- B.3 Benchmarking . . . 31

C Parameters and Performance **32**
- C.1 The parameters n, t, a and b . . . 32
- C.2 Bandwidth in the static-committee setting . . . 33
- C.3 Computation in the static-committee setting . . . 33
- C.4 Performance in the dynamic/proactive setting . . . 34
- C.5 Some numerical examples . . . 35

D Deploying in a Blockchain Environment **35**
- D.1 Committee Sizes . . . 36
- D.2 Uses of Additive Key Derivation . . . 37
- D.3 Recovery from Catastrophic Failures . . . 37

E The Full Agreement Protocol **38**
E.1 Optimizations and Variants 38

F Proofs of Theorems 1 and 2 **40**

G Security Proof of the Threshold Signature Protocol **41**
G.1 Centralized Schnorr 43
G.2 Threshold Schnorr for a single message 44
G.3 Threshold Schnorr for multiple messages 47
G.4 Threshold Schnorr with a Super-Invertible Matrix 48
G.5 Threshold Schnorr with Packing 49
G.6 Putting it All Together 49
G.7 Security for the Dynamic Setting 52

H Mixed Adversary Model and Dishonest Majority **53**

I Refreshing Packed Secrets **53**
I.1 Method One: Sharing One Polynomial 54
I.2 Method Two: Sharing a Polynomials 55

J The Parameter-Finding Utility **55**
J.1 Example Parameters 55

1 Introduction

In this work, we describe a suite of protocols that we call SPRINT[1], aimed at generating many Schnorr signatures at a low amortized cost. SPRINT consists of a single interactive distributed key generation (DKG) for generating message-independent ephemeral randomness, followed by a *non-interactive and robust* signature generation for many messages. Here, *robustness* means that with a sufficient number of honest parties, the protocol is guaranteed to output the requested signatures.

Threshold Schnorr signature schemes have seen a revival due to applications in the blockchain space. However, the bulk of existing work focuses on the case of a small number of signers, targeting applications such as key custody and multi-signatures. For those cases, one can afford a non-robust scheme where a single misbehaving party can cause the protocol to abort: If the misbehaving party can be identified, then it can be removed before re-running the protocol. This is indeed the approach most recent schemes embrace (e.g., [29, 9, 16, 30]). However, the remove-and-restart approach does not scale well with the number of signers, since the protocol may need to be restarted as many times as the number of misbehaving parties. Also, this approach cannot be used in a fully asynchronous setting, where there is no distinction between a malicious party that refuses to participate and an honest party that is just slow. Here, we study *robust* threshold Schnorr signatures in scenarios with many messages and many signers (possibly hundreds of them), in an asynchronous setting.

One of the motivating scenarios for considering a large set of signers signing many messages is provided by blockchain settings, where the validator nodes should generate signatures on behalf of the blockchain (see more below). That use case precludes non-robust protocols, as it requires an asynchronous protocol that remains feasible for many signers. At the same time, public blockchains provide tools such as a broadcast channel and PKI, which can simplify the design of the signature protocol. Moreover, the large number of parties makes it reasonable to assume a large honest majority, a significant advantage when building robust protocols.

Let us recall Schnorr-type signatures. They work over a group of prime order p with a generator G; a signature on a message M relative to secret key $s \in \mathbb{Z}_p$ and public key $S = s \cdot G$, has the form $(R, r + e \cdot s)$, where $r \in \mathbb{Z}_p$ is an ephemeral random secret, $R = r \cdot G$ is ephemeral randomness, and $e = \mathsf{Hash}(S, R, M) \in \mathbb{Z}_p$. A standard way to compute *robust* threshold Schnorr signatures among n parties who secret-share a long-term secret key s is to run a *distributed key generation (DKG)*[2] procedure [17] that produces a message-independent ephemeral randomness $R = r \cdot G$ where r is a fresh random value secret-shared among the parties. This phase is often called preprocessing or just DKG, and the message-independent ephemeral randomness is often called presignatures. Then, the parties use their shares of s and r to produce signature shares that can be combined into a single standard Schnorr signature. The bulk of the cost for signature generation is the DKG procedure that has $O(n^2)$ cost both in terms of bandwidth and computation.

Robust threshold Schnorr schemes have been known for over 20 years [40, 17], but they are less efficient than their non-robust counterparts. These robust protocols include at least 2–3 rounds to generate message-independent ephemeral randomness, and at least one additional round for signature generation. Moreover, the randomness-generation rounds are expensive, using a bandwidth of at least $\Omega(n^2)$ broadcasted group elements. Non-robust schemes can reduce the randomness generation part to a single round, performed before knowing the message to be signed, and a single

[1] SPRINT is a permuted acronym for "Robust Threshold Schnorr with Super-INvertible Packing".

[2] Throughout the paper, we use a DKG protocol for different purposes, including ephemeral Schnorr randomness generation, long-term key generation, and proactive refreshment.

non-interactive message-dependent round (where parties just output signature shares).

Our robust signature protocol features a two-round message-independent distributed ephemeral randomness generation, followed by a single non-interactive signature generation round. However, the latter non-interactive round can produce signatures for *many messages*, hence amortizing the cost of the randomness generation protocol over many signatures. The protocols we present can produce thousands of signatures in each run, at a communication cost similar to one execution of a synchronous non-robust protocol in prior work [17].

Our protocols are flexible: they are useful in the fixed-committee setting where the same set of parties is used repeatedly, but extends seamlessly to the dynamic/proactive setting where each run of the protocol is done by a different committee with refreshed shares of the secret key. They naturally support large systems, where committees are periodically sub-sampled from among the overall population of parties and the required shared secret state is transferred to the selected parties. The protocols are also *modular*: we present a high-level protocol based on a generic agreement protocol (for the parties to agree on a set of correctly dealt shares) instantiated on an asynchronous broadcast channel. Without tying the high-level signature protocol to the details of the agreement or the communication model, we are able to take advantage of systems (such as blockchain) that natively provide agreement and communication primitives.

This agreement protocol is instrumental in achieving one of our significant design goals, namely, to perform well in the optimistic case of normal network conditions, but also to avoid degrading performance unnecessarily when network delays (possibly adversarially induced) are significant. Crucial for ensuring this property is to achieve agreement as soon as possible among a sufficient number of parties. This calls for forgoing techniques such as complete secret sharing [34] where all honest parties must receive shares of the secret, hence adding longer delays (and latency) to the protocol completion.

We next describe techniques used to achieve the above functional and performance properties of our solution, starting with two main components: (a) an early agreement protocol allowing non-complete sharing and (b) "extreme packing" that combines packed secret sharing [14] with super-invertible matrices [26] to extend the number of signatures we get from a single ephemeral-randomness creation stage.

A simple early-agreement protocol. Many threshold systems require *complete secret sharing*, i.e., all honest parties must receive shares of the secret. This means that honest parties cannot terminate until they ensure that all other honest parties will eventually learn their shares. The completeness requirement often adds significant complexity to the protocol and an opportunity for the adversary to create high-latency executions in the asynchronous setting. In our protocols we forgo completeness and its adverse effects by only requiring that a sufficiently large subset of honest parties learn their shares so that they can generate signatures; there is no need to ensure that *all* honest parties get shares.

Weakening the completeness requirement of secret sharing allows us to use a very simple agreement protocol over the underlying asynchronous broadcast channel. Furthermore, the use of a broadcast channel enables *verifiable complaints* by shareholders, namely proofs that a dealer sent bad shares. Our use of these complaints is markedly different than in prior works. In protocols that aim at complete sharing (such as [20]), a party uses the complaints to inform other parties that it is missing its share, triggering a complex protocol by which the other honest shareholders help them get their missing shares. In contrast, we use the complaint to disqualify the bad dealer, there is no need to help the complaining shareholder get any more shares. This technique simplifies

the agreement protocol and saves rounds of broadcast[3] (see Sections 2.2 and 4 and Appendix E for details). We believe that this simple agreement protocol could find other uses beyond DKG and threshold signatures.

Extreme packing. To maximize efficiency, we introduce an efficiency parameter a, such that each run of the protocol produces $a(n-2t)$ signatures where t is the maximal number of corrupted parties supported by the protocol. In more detail, we use super-invertible matrices [26] to get a sharing of at least $n-2t$ random polynomials for every run of the ephemeral randomness generation, and use packed secret sharing [14] to put a random values in each of these polynomials (see Sections 2.4 and 2.5).[4]

We pay for this extreme packing with a slight reduction in resilience: To withstand t corrupted parties, the number of nodes that we need is $n \geq 3t + 2a - 1$, compared to $n \geq 3t + 1$ for a naive protocol that generates a single signature.[5] The result is a bandwidth-optimal protocol, up to some not-too-large constants: With n parties, it provides resilience against $\Omega(n)$ corrupted parties, using broadcast bandwidth of only $O(1)$ group-elements/scalars per signature, in *both the optimistic and the pessimistic cases* (where the number of faulty parties is small or large, respectively). We stress that the odds of everybody participating honestly diminishes as the number of parties grows, so in the large-committee setting it becomes more important to have an efficient pessimistic path. In our protocol, the pessimistic case features additional complaints, but those add at most $O(t/a)$ group-elements/scalars per signature.

For a few examples in the static-committee setting (and assuming no complaints), setting the efficiency parameter at $a = n/5$, they withstand $t = n/5$ corrupted parties and consume broadcast bandwidth of roughly 17.33 scalars/group-elements per signature. To support $t = n/4$ we must reduce the efficiency parameter to $a = n/8$, resulting in a per-signature bandwidth of about 34 scalars/group-elements. This $O(1)$ complexity is to be contrasted with the $O(n^2)$ complexity of the standard threshold Schnorr scheme [17]. See Appendix C.1.

1.1 Other Techniques

Achieving high efficiency requires the use of many ideas and techniques, beyond the two main ones that we described above. Below is a list of these techniques, in no particular order. See Section 2 for a detailed overview of the entire protocol and the roles that these techniques play.

Local SIMD computation. Working with packed secret sharing increases the number of secrets shared, but current MPC solutions for using packed secret sharing entail non-trivial protocols, even for simple functions [19]. For Schnorr signatures we need to compute $s \cdot (e_1, \ldots, e_a) + (r_1, \ldots, r_a)$ where s and the r_v's are secret and the e_v's are public. While simple, an MPC protocol for computing that function still seems to require interaction, since it includes a product. Furthermore, when using simple Shamir sharing for s, some joint processing is needed to create multiple signatures.

To enable a more efficient protocol with full advantage of packing and to avoid interaction, we introduce the following technique. We share the long-term secret key in a packed vector $(s, \ldots, s)$ instead of just the single scalar s. This enables SIMD generation of the partial signature, with each

[3] Our use of an underlying broadcast channel also obviates the need to find a biclique of dealers and shareholders, which is sometimes needed when giving up completeness, and which can be computationally hard (cf. [4]).

[4] We also describe some optimizations related to faster multiplication by super-invertible matrices in Section 2.4 and Appendix B.

[5] Since our techniques apply in the asynchronous setting, they inherently require $n \geq 3t + 1$; see Appendix H.

party using only a local multiplication (without degree reduction), with randomization done locally as well. Using this technique, signature generation becomes non-interactive: The only communication required is for the party to broadcast their partial signature, after which anyone can assemble the signatures themselves. The cost is a reduction in the resilience to $t < (n - 2a + 2)/3$. See Section 2.6 for details.

Refreshing packed secrets. In the dynamic/proactive setting, we need to refresh the sharing of the packed vector $(s, \ldots, s)$. This requires a generalization to the GRR protocol [18], see Section 2.7 and Appendix I.We remark that in the current version of the writeup we only prove security of the static protocol. The proof for the dynamic/proactive protocol should be a fairly straightforward extension, using the same techniques. See a brief discussion in Appendix G.7.

Security of distributed parallel Schnorr signatures. The starting point for our protocol is similar (though not quite identical) to the GJKR distributed Schnorr signature protocol from [17], which we extend and optimize to sign many messages. However, GJKR-like protocols [17] are known to fail in the concurrent setting where the protocol is run in parallel for multiple messages; specifically, such protocols are open to ROS-type attacks [12, 6]. Our work focuses on signing a given set of messages (a batch) in parallel. To enable this parallelism and avoid ROS-type attacks, we use a mitigation technique similar to prior work (e.g., [29, 21]). As far as we know, prior to our work this specific technique was only analyzed in the generic group model for ECDSA signatures [21]. In our case, we show it is sufficient for proving the security of our protocols (for signing a single batch of messages) via reduction to the hardness of the discrete logarithm problem in the programmable random-oracle model. See Section 2.3 and Appendix G. These techniques do not guarantee concurrent security for signing multiple batches in parallel. For this, Shoup [39] shows that technique from FROST can be combined with our protocols to obtain full concurrent security (see detailed discussion on this in Section 1.3).

Robust threshold signatures. Our protocols provide robustness in a strong sense. They terminate with signatures for all $a(n - 2t)$ input messages as soon as $t + 2a - 2$ honest parties output their shares. Invalid shares can be identified based on public information and discarded. This holds in both synchronous and asynchronous networks. In the former case, after two rounds of broadcast for generating ephemeral randomness, parties generate *non-interactively* the shares from which all signatures are recovered.

Smaller sub-sampled committees using a beacon. To use our protocols in massive systems with a huge number of nodes, one needs some mechanism to sub-sample the committees from among all the nodes in the system. One natural approach is to use self-selection via verifiable random functions (VRFs), as done, e.g., in [8]. However, this results in somewhat loose tail bounds and thus somewhat-too-big committees.

Instead, we note that we can get smaller committees by using a randomness beacon to implement the sub-sampling, resulting in better bounds and smaller committees. Thus, when acting in this large dynamic committee settings, we augment the signature protocol to implement this beacon, which turns out to be almost for free in our case. See Section 2.8 for more details. See also Appendix A for an additional optimization in this setting: using smaller optimistic parameters by default with a safe fallback mechanism to pessimistic parameters.

1.2 Prior Work

Recent years saw a lot of activity trying to improve the efficiency of threshold signature schemes, including underlying techniques such as verifiable secret sharing (VSS) and distributed key generation (DKG), much of which focused on asynchronous protocols and some emphasizing robustness (guaranteed output delivery). Below we focus on some of the more recent works on these subjects.

Threshold Signatures. Komlo and Goldberg described FROST [29], a non-robust threshold Schnorr signature protocol that requires a single-round signing protocol after a single-round preprocessing phase. The improved round complexity comes at the expense of robustness, as it uses additive sharings and requires correct participation of all prescribed signers. In our case, we use two rounds of interaction in a message-independent phase but can then generate multiple signatures non-interactively and with guaranteed output delivery. Our schemes are designed to work in an asynchronous regime hence requiring a super-majority of honest parties (see details in Appendix H).

ROAST [38] presents a wrapper technique that can transform concurrently secure non-robust threshold signature schemes with a single signing round and identifiable abort into a protocol with the same properties but also robust in the asynchronous model. In particular, this applies to the FROST protocol resulting in a scheme with concurrent security for any threshold $t < n$ and optimal robustness for up to $n-t$ parties. The price for this strengthening is significant: it involves $O(tn^2 + tn\lambda)$ per-signature transmitted bits (λ is a security parameter) assuming a trusted coordinator and $O(tn^3 + tn^2\lambda)$ without the coordinator; whereas we only require $O(\lambda)$ broadcasted bits (strictly better even when considering a quadratic overhead of the underlying broadcast).

Garillot et al. [16] implement a threshold Schnorr signature based on deterministic signing, e.g., EdDSA, in order to avoid the potential risks of randomness reuse. They present a dishonest-majority non-robust scheme using zero-knowledge proof and garbling techniques that, while optimized for this specific application, is much more expensive than protocols that do not offer deterministic signing (like FROST and our SPRINT protocols).

Lindell [30] presents a threshold Schnorr signature scheme proven under standard assumptions in the UC model. The focus of that work was conceptual simplicity and UC security rather than optimal efficiency. As in FROST, it utilizes additive sharing, hence necessitating the choice of a new set of signers when a chosen set fails to generate a signature.

For ECDSA signatures, Groth and Shoup [20] recently described a rather efficient ECDSA signing protocol, with emphasis on guaranteed output delivery over asynchronous channels. (The underlying VSS in their work achieves completeness, which is not needed in our case.) They use verifiable complaints in order to notify other parties that they do not have a share. These complaints trigger a complex protocol, by which honest shareholders help each other to get all their missing shares.

Joshi et al. [27] address the lack of concurrent security in the basic threshold Schnorr scheme from [17] by running two DKG executions per signature and using a mitigation technique similar to the one we use here to bind a batch of messages to be signed. However, while our solution generates multiple signatures with a single DKG run, theirs requires two such runs per single signed message.

Distributed Randomness Generation (DKG).[6] As we said, a key distinction between our work and previous DKG protocols in the signature setting [32, 42, 1], is that we *do not require complete sharing* (where all honest parties must receive their shares). While completeness may be desired

[6] Recall we use DKG to refer to distributed key generation for long-term keys, for generating ephemeral randomness as needed in Schnorr signatures, and also for proactive refreshment.

in traditional MPC applications, eschewing this requirement is not a weakness but a feature in our case, as it enables more efficient signature protocols.

Neji et al. [32] design a DKG intended to avoid the need to reveal the shares of inactive (or slow) shareholders for disqualification as required in the GJKR [17] solution. However, they do so by requiring additional rounds of interaction and significant extra computational cost, namely the party who gets complained does $O(n)$ group additions and each other party does $O(t)$ scalar multiplications where t is the corruption threshold (these costs are merely for handling complaints beyond the verification). We achieve higher performance by using publicly verifiable complaints: in our protocols, each party can verify that a complaint is valid by doing a constant number of group operations and without any additional interaction.

Yurek et al. [42] described a randomness generation protocol over asynchronous communication channels, in the context of the offline phase of generic secure MPC. They provide completeness for secret sharing needed for their MPC applications. As in a recent work by Groth and Shoup [20], they use verifiable complaints, yet unlike our work, they do not disqualify dealers upon a verifiable complaint—they instead complete the set of shares. Their asynchronous VSS has an amortized network bandwidth $O(n \log n)$ in the optimistic case and $O(n^2 \log n)$ in the pessimistic case.

Abraham et al. describe Bingo [1], a packed method for asynchronous secret sharing that allows a dealer to generate many sharings at an amortized communication cost of $O(\lambda n)$ per secret. This solution requires KZG-style polynomial commitments [28] to get completeness (and thus relies on pairing-friendly groups). Specifically, the dealer performs a KZG commitment to a polynomial of degree $2t$ (where $n = 3t + 1$), which concretely is slightly more expensive than our protocol. Also, our agreement sub-protocol makes a more direct usage of the underlying broadcast channel than the agreement in Bingo, and is more efficient.

Various other papers (e.g., [10, 11]) deal with the question of asynchronous DKG. However, they do not directly relate to our paper as the main thrust of their work is reaching an agreement in the asynchronous setting. In contrast, we assume an underlying broadcast channel, simplifying the agreement significantly.

1.3 Subsequent Work

There have been several papers published after our paper was first made public.

Shoup's Many Faces of Schnorr. In [39], Shoup presents a unifying framework for obtaining robust concurrently-secure threshold Schnorr signatures combining techniques from our work and FROST [29]. This framework applies to two-phase protocols, like ours, consisting of an offline phase for generating "presignatures" (a.k.a., ephemeral randomness), and then an online phase for generating signatures from those presignatures. The concurrent-security aspect of these protocols means that many copies of the online phase can be run concurrently, as long as sufficiently many unused presignatures are available. Shoup shows that concurrent security can be added to any protocol within this framework (including ours) in one of two ways: either using two fresh DKG-generated secret sharing of ephemeral randomness à-la-FROST (thus doubling the cost), or using a randomness beacon (which adds rounds of communication).

Groth-Shoup Asynchronous Robust DKG. In [22], Groth and Shoup present an asynchronous robust DKG protocol which can be used as a basis for a threshold signature protocol, that require a total of $O(n\lambda)$ bits of point-to-point communication per signature over the optimistic path (roughly

when all parties behave honestly), amortized over $O(n^2)$ signatures. The optimistic path communication complexity matches (asymptotically) our communication complexity of $O(\lambda)$ bits broadcast per signature.[7]

However, the Groth-Shoup protocol is a lot less efficient on the pessimistic path, when parties misbehave: Its communication complexity increases by a factor $O(t')$ where t' is the number of actual misbehaving parties. In contrast, the communication complexity of our protocol increases by at most a small constant factor, no matter how many parties misbehave (as long as there are at most t of them). On the other hand, the Groth-Shoup protocol can withstand up to $(n-1)/3$ misbehaving parties, compared to our $t \leq (n-2a+1)/3$. Our protocol is therefore a better choice in the large-committee setting, where consistent performance also on the pessimistic path is important, and where it is reasonable to assume a larger honest majority. The Groth-Shoup protocol may be better in the small-committee setting, where higher resilience is more important and assuming the optimistic path makes more sense.

The main difference between our protocol and Groth-Shoup stems from the fact that the latter requires complete secret sharing, where all the honest parties get their shares. In particular, if a dealer misbehaves and does not appropriately distribute shares to some honest parties, these honest parties need other honest parties to help them reconstruct their shares, whereas our protocol just disqualifies that dealer. On the other hand, the Groth-Shoup protocol uses complete secret sharing to eliminate the need for polynomial commitments in the sharing phase. Instead, they use error correction to reconstruct signature shares at the end of the protocol without having to check validity against some public commitment.

Another difference is that [22] uses a construction based on Pascal triangle for super-invertible matrices, which is better than the small Vandermonde construction we use in Appendix B. This way, they reduce the cost of evaluating the product by the super-invertible matrix from $\approx (b-1)n \log n / \log p$ scalar-element products in that solution to $\approx b(n-(b+1)/2)+1$ group additions (which correspond to about $(b(n-(b+1)/2)+1)/(1.5 \log p)$ scalar-element products). Our proposal to use the ECFFT-EXTEND algorithm (see Section 2.4) is more efficient asymptotically ($O(k \log k)$ scalar-element products, for $k = \max(b, n-b)$) but the Pascal solution would most likely perform better up to $n \approx 8000$.

1.4 Organization

The rest of this manuscript is organized as follows: In Section 2, we provide a high-level step-by-step overview of our protocols and the various components that are used in them. In Section 3, we describe in more detail our high-level protocol for the static (fixed-committee) and dynamic settings. In Section 4, we describe the basic agreement protocol that we use in the static-committee setting, the agreement in the dynamic setting can be found in Appendix E. Security proofs and additional details are deferred to appendices. In particular, in Appendix D, we discuss how to use SPRINT in one of our motivating applications to implement a large-scale signature service over a public blockchain.

[7] Broadcasting messages of size $\ell \geq n\lambda$ bits, as done in our protocol, can be achieved using a total point-to-point communication of $O(\ell n)$ bits [15, 31].

2 Technical Overview

We consider a static setting where the set of parties (a shareholder committee) is fixed and a dynamic one where shareholder committees change over time while keeping the system's signing key (in particular, its public verification key) unchanged. In the latter case, shares are refreshed and proactivized between committees. We begin by describing our protocols in the static committee setting, and discuss only towards the end the extra components for the dynamic/proactive settings. The basic protocols for these two settings are shown in Figs. 1 and 2.

In the static case, we have a committee that holds shares of the long-term secret key s, shared via a degree-d polynomial $\mathbf{F}(X)$ with party i holding $\sigma_i = \mathbf{F}(i)$ (for some degree d that we determine later) and where $s = \mathbf{F}(0)$. They first run a distributed key-generation (DKG) protocol to generate a sharing of ephemeral randomness, then use their shares of the long-term secret and ephemeral randomness to generate Schnorr-type signatures on messages. The DKG and signature protocols can be pipelined, where the committee uses the randomness that was received in the previous run to sign messages, and at the same time prepares the randomness for the next run.

While the static setting features just a single committee, we still often refer to parties as *dealers* when they share secrets to others, and as *shareholders* when they receive those shares. In the dynamic setting, these will indeed be different parties, but in the static case, they may be the same.

Notations. We use Greek letters (e.g., σ, ρ, π, ϕ) and lowercase English letters (e.g., e, r, s) to denote scalars in $\mathbb{Z}_p$, and also use some English lowercase letters to denote indexes (i, j, k, ℓ, u, v) and parameters (a, b, n, t). We denote the set of integers from x to y (inclusive) by $[x, y]$, and also denote $[x] = [1, x]$. We rely on a group of prime order p generated by G. We use the additive notation for this group. Group elements are denoted by uppercase English letters (G, S, R, etc.). Polynomials are denoted by bold Uppercase English letters ($\mathbf{F}, \mathbf{H}, \mathbf{I}, \mathbf{Y}, \mathbf{Z}$), and commitments to them are sometimes denoted with a hat ($\hat{\mathbf{F}}, \hat{\mathbf{H}}$).

2.1 Starting Point: The GJKR Protocol

Our starting point is the protocol of Gennaro et al. [17] for distributed key generation (DKG), and a variation on their use of that protocol for Schnorr signatures. In their DKG protocol, each dealer uses Verifiable Secret Sharing (VSS) to share a random value; parties then add all the shares from dealers that shared their values correctly (thus requiring an agreement protocol on which dealers fall in this set, denoted QUAL). Specifically, each dealer D_i shares a random ephemeral secret (which is later used to compute ephemeral randomness and partial signatures) using a degree-d' polynomial $\mathbf{H}_i$ (for some degree d' that we define later), and commits publicly to this polynomial. Concretely, D_i shares the random ephemeral secret $\mathbf{H}_i(0)$ by sending shares $\mathbf{H}_i(j)$ to each shareholder P_j.

The shareholders then agree on a set QUAL of "qualified dealers" whose values will be used, and a corresponding shareholder set HOLD that were able to receive valid shares. Shareholders in HOLD can compute shares for the ephemeral secrets from the shares that they received from these qualified dealers. Namely, each shareholder can add the shares (i.e., the Shamir shares of ephemeral secrets of dealers) that they received from dealers in QUAL, and the resulting ephemeral secret is shared via the polynomial $\mathbf{H} = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i$.

In our protocol, shareholders use their shares on polynomials $\mathbf{H}$ (the ephemeral secret) and $\mathbf{F}$ (the long-term secret) to compute Shamir shares of the signatures, and then reconstruct the signatures themselves. We note that this is somewhat different from the signature protocol in [17]: there, it is

the dealers in QUAL that generate the signature (and HOLD is only used as a backup to reconstruct the input of misbehaving dealers), whereas we let the shareholders in HOLD generate the signature directly. Our variant could be more round-efficient in some cases, and is easier to deploy in a proactive setting where the long-term key is shared using Shamir sharing (as opposed to additive sharing as used in the GJKR protocol). But otherwise these protocols are very similar.

Pedersen vs. Feldman Commitments. It was pointed out by Gennaro et al. [17] that sharing randomness usually requires the dealers to commit to their sharing polynomials using statistically-hiding commitments such as Pedersen's [35]. Using the less expensive Feldman secret sharing, where dealers commit to coefficients h_{ij} of their polynomials by broadcasting the group elements $h_{ij} \cdot G$, are susceptible to rushing attacks in the DKG setting. Luckily, Gennaro et al. prove in [17, Sec 5] that for the purpose of generating the ephemeral randomness for Schnorr signatures, it is safe to use Feldman secret-sharing, and their proof techniques extend to our signature protocol as well.

We note that for efficiency reasons, in our protocols we use commitments to the value of the polynomials at certain evaluation points rather than to the coefficients as done in [17] (see Appendix A).

2.2 The Agreement Protocol

We utilize the QUAL-agreement protocol in two different settings: for generation of ephemeral randomness (in both the static and dynamic setting), and for re-sharing of the long-term key (in the dynamic setting only). We observe that randomness generation is less demanding of the agreement protocol than key-refresh: For key-refresh we need the shareholders to have shares from at least $d+1$ dealers (d is the degree of the sharing polynomials), whereas randomness generation can work even with a single honest dealer. Therefore, in the static setting we use a weaker (and more efficient) agreement protocol than in the dynamic setting. Both protocols use PKI, and both operate over a total-order (aka atomic) broadcast channel, providing eventual delivery of messages from honest parties, sender authentication, and prefix consistency (i.e., the views of any two honest parties are such that one is a prefix of the other).

We start with the more efficient (but weaker) protocol for the static setting. The protocol begins with the dealers distributing their shares, and then the shareholders engage in a protocol to agree on sets of "qualified" and "bad" dealers $\mathsf{QUAL}, \mathsf{BAD}$, and a set of shareholders HOLD. We want the following properties: (i) every shareholder in HOLD received valid shares from every dealer in QUAL, and (ii) BAD consists entirely of dishonest dealers. This protocol is parameterized by d_0, d_1 (to be defined later as a function of the number of corrupt parties and some additional parameters), and it ensures that $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$.

In more detail, each dealer D_i broadcasts all the shares that it deals, encrypted under the keys of their intended recipients, together with commitments to the sharing polynomial $\mathbf{H}_i$. As this information is visible to all, shareholders that receive shares that are inconsistent with the commitments can broadcast a *verifiable complaint* against a dealer, consisting of a proof that the dealer has sent them a bad share.

The shareholders initially set QUAL to the first d_1 dealers whose messages appeared on the broadcast channel. Then each shareholder broadcasts verifiable complaints if they have any, and otherwise they broadcast the empty set (signifying that they have all the shares from dealers in QUAL). Now, QUAL contracts by eliminating all the dealers who have a valid verifiable complaint against them on the broadcast channel, moving them to the set BAD. The set HOLD is fixed to the

first d_0 shareholders who broadcast verifiable complaints (or the empty set) that were verified as valid complaints. By construction, we have $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}|+|\mathsf{BAD}| \geq d_1$, and the set BAD contains only (verifiably) dishonest dealers. Also, since $\mathsf{QUAL}, \mathsf{BAD}$, and HOLD are determined by what is visible on the broadcast channel, then all honest shareholders that read up to some point in the channel will agree on these sets. This protocol's specification can be found in Fig. 3, and the proof is provided in Theorem 1.

In the dynamic setting (that includes also key-refresh), we need to ensure $|\mathsf{QUAL}| \geq d_1$ (as opposed to just $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$). To that end, we run iterations of the basic protocol above. At the beginning of the $i+1$'st iteration, we add to QUAL as many new dealers as the number of dealers that were added to BAD in the i'th iteration. Once no more dealers are added to BAD, we have $|\mathsf{QUAL}| \geq d_1$, and we are done. A full specification is found in Fig. 5 in Appendix E, with a proof in Theorem 2.

2.3 Signing Many Messages in Parallel

Our large-scale signature service needs to handle signing many messages in parallel, which brings up a security problem: The proof of security from [17, Sec 5] when using Feldman commitments for Schnorr signatures, requires that the reduction algorithm makes a guess about which random oracle query the adversary intends to use for the signature. When signing many messages in parallel, the reduction will need to guess one random-oracle query per message, leading to exponential security loss. Moreover, Benhamouda et al. demonstrated in [6] that this is not just a problem with the reduction, indeed this protocol is vulnerable to an actual forgery attack when many messages are signed in parallel. To fix this problem, we use a mitigation technique somewhat similar to [29, 21], where the ephemeral secrets are all "shifted" by a public random value δ, which is only determined after all the messages and commitments are known.

As recalled in the introduction, a Schnorr signature on a message M^v relative to secret key s and public key $S = s \cdot G$, has the form $(R^v, r^v + e^v \cdot s)$, where r^v is an ephemeral random secret, $R^v = r^v \cdot G$, and $e^v = \mathsf{Hash}(S, R^v, M^v)$, where Hash maps arbitrary strings into $\mathbb{Z}_p$. (We are using a superscript v to indicate a plurality of messages and their respective signatures.) In our context, we first run DKG to generate all the required r^v's and corresponding R^v's, and get from the calling application all the messages M^v's to be signed. Then we compute $\delta = \mathsf{Hash}(S, (R^1, M^1), (R^2, M^2), \ldots)$ and $\Delta = \delta \cdot G$. The signature on M^v is then set as $(R^v + \Delta, r^v + \delta + e^v \cdot s)$, where $e^v = \mathsf{Hash}(S, R^v + \Delta, M^v)$.

With this mitigation technique, the reduction only needs to guess the random-oracle query in which δ is computed, recovering the argument from [17, Sec 5] and reducing security to the hardness of computing discrete logarithms in the random-oracle model. See Appendix G.3. We note that our specific mitigation techniques provide security for *a single run of the protocol* on input a set of multiple messages to be signed, but it does not imply concurrent security for multiple parallel runs of the protocol on different sets of messages. Following [39], we can obtain concurrent security by either adopting the FROST mitigation (that requires doubling the DKG cost) or by relying on a beacon (which would add one broadcast round).

2.4 Using Super-Invertible Matrices

As described so far, we would need to run a separate copy of the DKG protocol to generate each ephemeral secret r^v, but we can do much better. For starters, assume that we can ensure many

honest dealers in the set QUAL (say at least b of them). Then we can use a (public) super-invertible matrix [26] to generate b random ephemeral values in each run of the protocol.

Recall that the DKG protocol has each dealer D_i share a random polynomial $\mathbf{H}_i$, then the shareholders compute a single random polynomial $\mathbf{H}' = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i$ and the ephemeral random secret is $\mathbf{H}'(0)$. Intuitively, the polynomial $\mathbf{H}'$ is random if even a single $\mathbf{H}_i$ is random, so a single honest dealer in QUAL is enough to get a random ephemeral value. But if we have many honest dealers in QUAL, then we can get many random polynomials. Specifically, suppose we have b honest dealers in QUAL and let $\Psi = [\psi_i^u]$ be a b-by-n super-invertible matrix, i.e., each b-by-b sub-matrix of Ψ is invertible. Then we still have each dealer D_i share just a single polynomial $\mathbf{H}_i$, but now the shareholders can construct b random polynomials $\mathbf{H}^1, \ldots, \mathbf{H}^b$, by setting $\mathbf{H}^u = \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i$ for all $u \in [b]$. By the same reasoning as before, if we have b honest dealers in QUAL with random input polynomials $\mathbf{H}_i$, then the b output polynomials will also be random and independent since the b-by-b matrix corresponding to the rows of these b honest dealers is invertible.

The actual proof is more involved since we still use Feldman commitments in the protocol, which means that a rushing adversary can bias the output polynomials somewhat. But using essentially the same reduction as before, we can still reduce the security of the Schnorr signature protocol to the hardness of computing discrete logarithms in the random oracle model. One technical point is that the security proof in the asynchronous communication model requires that the set QUAL is included in the hash function query that determines δ. That is, we compute $\delta = \mathsf{Hash}(S, \mathsf{QUAL}, (R^1, M^1), (R^2, M^2), \ldots)$. The reason is that in the asynchronous case, we cannot guarantee that all honest dealers will be included in QUAL. If we didn't include it in the hash query, then the simulator would have to guess the set QUAL, incurring at least an $\binom{n}{b}$ loss factor in security.

We note that to ensure b honest dealers in QUAL, it is enough to run the "weaker" agreement protocol (Fig. 3) with $d_1 = b + t$, where t is an upper bound on the number of dishonest dealers. Indeed, that protocol ensures that $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1 = b + t$, and BAD contains only dishonest dealers. Therefore, the number of dishonest dealers in QUAL is at most $t - |\mathsf{BAD}|$, and the number of honest dealers is at least $|\mathsf{QUAL}| - (t - |\mathsf{BAD}|) = |\mathsf{QUAL}| + |\mathsf{BAD}| - t = d_1 - t = b$.

Faster Multiplication by a Super-Invertible Matrix. While the use of super-invertible matrices enables us to produce many more random shared secrets without increasing bandwidth, computing all these sharings requires that each shareholder multiply their sub-shares by that super-invertible matrix "in the exponent".[8] The super-invertible matrix multiplication is the most computationally intensive operation in the protocol. We thus should carefully implement the matrix multiplication to have good computational efficiency in practice.

We propose two solutions to make these operations more efficient. The first solution, pointed out to us by Victor Shoup, is to use a Vandermonde matrix Ψ corresponding to the powers of small scalars. We show in Appendix B.1, that a variant of the Horner's rule allows to evaluate the multiply-by-Ψ operation using $(b-1)n$ scalar-by-element products with $\log n$-bit scalars (instead of full-length scalars, that is $\log p$-bit scalars). This is equivalent to about $(b-1)n \log n / \log p$ full scalar-by-element product, that is a $\log n / \log p$ speed-up over the naive solution. In practice, p has at least 256 bits, while $n = b + t$ varies but is unlikely to be higher than 10 bits, so this is a more than 25× speed-up.

Our second solution is new and consists of selecting Ψ so that it corresponds to FFT-related operations. However, when implementing Schnorr signatures over the elliptic curve ED25519, the

[8] We use additive notation for group operations, but sometimes use the traditional exponentiation terminology.

scalar field $\mathbb{Z}_p$ does not even have a 2^3-th root of unity.[9] Instead, we show that we can use the ECFFT-EXTEND algorithm from Ben-Sasson et al. [5], resulting in $O(k \log k)$ scalar-by-element products, where $k = \max(b, t)$ and $n = b + t$. This is asymptotically better than the first solution. Details are provided in Appendix B.2.

We implemented both solutions, benchmarked them, and report results in Appendix B.3. In short, for ED25519, when $b = t$ is a power of 2, the small Vandermonde matrix solution is better in practice for up to $b = t = 2^8 = 256$, after which the ECFFT solution is more efficient.[10] The benchmarking code is available from https://github.com/fabrice102/ecfft-group, under the MIT license. This code is based on the code [7] and adapts it to work with polynomials with coefficients in a group, instead of in the base field.

2.5 Using Packed Secret Sharing

Similarly to above, we can also assume many honest parties among the set HOLD of shareholders, and use packed secret sharing [14] to get even more ephemeral shared values: If HOLD contains at least $2t + a$ shareholders (for some $a \geq 1$), then we can let each shared polynomial pack a values rather than just one: Each shared polynomial $\mathbf{H}^u$ will have degree $d' \geq t+a-1$ (rather than $d' = t$) and will encode the a values $\mathbf{H}^u(0), \mathbf{H}^u(-1), \ldots, \mathbf{H}^u(-a+1)$. (Below we denote these scalar values by $r^{u,v} = \mathbf{H}^u(1-v)$, with the corresponding group elements $R^{u,v} = r^{u,v} \cdot G$.)

Importantly, this amplifies the effect of using super-invertible matrices: We have each dealer D_i sharing a single random polynomial $\mathbf{H}_i$ of degree d', packing a values, and we derive b random degree-d' polynomials $\mathbf{H}^u$ from these sharings, which gives us $a \cdot b$ shared random scalars.

2.6 More Efficient Signing

Once the ephemeral secrets are shared, we use them—together with the shared long-term secret key—to generate many signatures. Computing on the packed ephemeral secrets would generically require a full-blown secure-MPC protocol among the shareholders, but we observe that we can generate all the a signatures from each packed random polynomial with only a single share-reconstruction operation.

To see how, recall again that a Schnorr-type signature has the form $(R^v, r^v + e^v \cdot s)$.[11] Our shareholders hold Shamir sharings of the secret key s and the vector $(r^1, r^2, \ldots, r^a)$ of ephemeral secrets (where $r^v = \mathbf{H}(1-v)$ for $v \in [a]$). Also, the public key S, the messages M^v's, and the group elements R^v's are publicly known, so everyone can compute all the scalars $e^v = \mathsf{Hash}(S, R^v, M^v)$. To improve efficiency, we also share the long-term key s in a packed form, namely the shareholders hold a Shamir sharing of the vector $(s, s, \ldots, s)$, via a polynomial $\mathbf{F}$ of degree $d = t + a - 1$ (i.e., $\mathbf{F}(1-v) = s$ for $v \in [a]$). All they need to do, therefore, is compute the pointwise linear function $(r^1, r^2, \ldots, r^a) + (e^1, e^2, \ldots, e^a) \odot (s, s, \ldots, s)$.

While pointwise addition can be computed locally, computing the pointwise product $(e^1, e^2, \ldots, e^a) \odot (s, s, \ldots, s)$ seems like still requiring a nontrivial interactive protocol, even for a known vector of e^v's. But we can eliminate even this little interaction, by assuming a larger honest majority and

[9] $p = 2^{252} + 27742317777372353535851937790883648493$ and the factorization of $p - 1$ is $2^2 \times 3 \times 11 \times 19821142323093075401308452576369 7 \times 27660262428164223993721868055713982666874 7$.

[10] The ECFFT solution performs better for $b = t$ is a power of two. But we show in Appendix B.2 that it also works for general b and t, with a cost depending on the smallest power of 2 larger or equal to $\max(b, t)$.

[11] We suppress here the index u, which is irrelevant for this discussion.

using higher-degree polynomials for the ephemeral randomness. Specifically, we assume that HOLD contains at least $2t+2a-1$ shareholders (so at least $t+2a-1$ honest ones), and modify the DKG protocol so that the sharing of the ephemeral secrets is done with random polynomials of degree $d' = d+a-1 = t+2a-2$ (rather than degree $t+a-1$).

Since the e^v's are known, each shareholder can interpolate the unique degree-$(a-1)$ polynomial that packs the vector $(e^1,\ldots,e^a)$. Denote this polynomial as $\mathbf{Z}$ (we have $\mathbf{Z}(1-v) = e^v$ for $v \in [a]$). Then each shareholder j with a share $\sigma_j = \mathbf{F}(j)$ for the long-term secret, can *locally* compute $\sigma'_j = \mathbf{Z}(j)\cdot\sigma_j$. Note now that the σ'_j's lie on the polynomial $\mathbf{Z}\cdot\mathbf{F}$ of degree $d+a-1$ that packs the vector $(e^1\cdot s,\ldots,e^a\cdot s)$, since $(\mathbf{Z}\cdot\mathbf{F})(1-v) = e^v\cdot s$ for $v\in[a]$.

Each shareholder j, with share ρ_j on an ephemeral-randomness polynomial, computes and broadcasts $\pi_j = \sigma'_j + \rho_j$, and we note that these π_j's lie on a polynomial of degree d' that packs all the values $(r^1+e^1 s,\ldots,r^a+e^a s)$. Moreover, if the ephemeral secrets were shared via a *random* degree-d' polynomial, then the π_j's constitute *a random sharing* of that vector. After seeing $d'+1 = t+2a-1$ valid shares of these broadcast values, everyone can reconstruct the polynomial and read out all the scalars $\phi^v = r^v + e^v\cdot s$ that are needed for these a signatures.

2.7 The Dynamic Setting

So far, we have described our protocols for the static (fixed committee) setting. Here we present the additional components that we need in the dynamic case, where we have different committees for the dealers and shareholders. Importantly, in all the protocols above we never assumed that the dealers and shareholders are the same committee, so they all still work as-is also in the dynamic setting. What is missing is a share-refresh protocol where the dealers can pass to the shareholders a sharing of the long-term secret s. Here we essentially just use the GRR protocols of Gennaro et al. from [18], with a minor adaptation since we need to share it in a packed manner.[12]

Each dealer D_i begins with a share σ_i of the long-term secret key s, shared using a "packed" polynomial $\mathbf{F}(X)$ of degree $d = t+a-1$. Namely, $\sigma_i = \mathbf{F}(i)$, and $\mathbf{F}(0) = \mathbf{F}(-1) = \cdots = \mathbf{F}(1-a) = s$. In addition, everyone knows a commitment to $\mathbf{F}$. D_i reshares its share using a fresh random degree-d polynomial $\mathbf{F}_i$ with $\mathbf{F}_i(0) = \mathbf{F}_i(-1) = \cdots = \mathbf{F}_i(1-a) = \sigma_i$, and also commits publicly to $\mathbf{F}_i$.

This is done in parallel to the sharing of the random, degree-d', polynomial $\mathbf{H}_i$. The shareholders then engage in an agreement protocol (cf. Fig. 5) to determine the sets HOLD of shareholders, $\mathsf{QUAL}_1, \mathsf{BAD}_1$ for the $\mathbf{H}$ dealers, and $\mathsf{QUAL}_2, \mathsf{BAD}_2$ for the $\mathbf{F}$ dealers, with $|\mathsf{HOLD}| \geq n-t$, $|\mathsf{QUAL}_1| \geq n-t$, and $|\mathsf{QUAL}_2| \geq d+1$.[13] Having received $\sigma_{ij} = \mathbf{F}_i(j)$ from each dealer $D_i \in \mathsf{QUAL}_2$, P_j then computes their share of the long-term secret as $\sigma'_j = \sum_{i\in\mathsf{QUAL}_2} \lambda_i\sigma_{ij}$. The λ_i's are the Lagrange coefficients for recovering $Q(0)$ from $\{Q(i) : i \in \mathsf{QUAL}_2\}$ for degree-d polynomials Q. As usual, denoting $\mathbf{F}' = \sum_{i\in\mathsf{QUAL}_2}\lambda_i\mathbf{F}_i$, the shares of shareholders in HOLD satisfy $\sigma'_j = \mathbf{F}'(j)$, and also

$$\mathbf{F}'(0) = \sum_{i\in\mathsf{QUAL}_2} \lambda_i\mathbf{F}_i(0) = \sum_{i\in\mathsf{QUAL}_2} \lambda_i\mathbf{F}(i) = \mathbf{F}(0).$$

Moreover, since all the $\mathbf{F}_i$'s satisfy $\mathbf{F}_i(0) = \mathbf{F}_i(-1) = \cdots = \mathbf{F}_i(1-a)$, then so does $\mathbf{F}'$.

[12] As described here, the protocol only works for resharing a packed vector of the form $(s,s,\ldots,s)$. But it is not very hard to extend it to reshare arbitrary packed vectors (using somewhat higher-degree polynomials), see Appendix I.

[13] Recall that in the dynamic setting we use an agreement protocol that provides stronger guarantees about the size of QUAL, than in the static setting. Namely $|\mathsf{QUAL}| \geq d_1$ instead of just $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$. See Section 2.2.

2.8 Sub-sampling the Committees

One of the main use cases for our protocol is an open system (such as a public blockchain), which could be very large. In this use case, the committees in each epoch must be sub-sampled from the entire population, and be large enough to ensure a sufficiently large honest majority with overwhelming probability.

One way of implementing this sub-sampling would be to use verifiable random functions (VRFs), but this would result in rather loose tail bounds and large committees. We can get smaller committees by having the committees implement also a randomness beacon, outputting a (pseudo)random value that the adversary cannot influence at the end of each run of the protocol. At the beginning of the $T+1$'st protocol, everyone therefore knows the value U_T that was produced by the beacon in the T'th run. Members of the $T+1$'st committee determine the members of the $T+2$'nd committee by applying a PRG on U_T.

To see why this helps, note that when the total population is very large, the number of honest parties in a committee chosen by VRFs is approximated by a Poisson random variable with parameter $\lambda = (1-f)n$, where f is the fraction of faulty parties in the overall population (and n is the expected committee size). On the other hand, the number of honest parties in a committee when using the randomness beacon follows a Binomial distribution with parameters $n, p = 1-f$. The Binomial turns out to be much more concentrated than the Poisson, hence the number of honest parties is much closer to $(1-f)n$ with the beacon than with the VRF.

Implementing the randomness beacon for our protocol turns out to be very easy. Since the T'th committee held a sharing of the long-term secret scalar s, they could locally compute a "sharing in the exponent" of $s \cdot \mathsf{Hash}'(T)$ (with Hash' hashing into the group). Namely, everyone computes the group element $E = \mathsf{Hash}'(T)$, then each dealer D_i in the T'th committee with share σ_i can compute and broadcast $U_{T,i} = \sigma_i \cdot E$, together with a Fiat-Shamir zero-knowledge proof that $U_{T,i}$ is consistent with the (public) Feldman commitment of σ_i (which is a proof of equality of discrete logarithms).[14] Once the qualified set QUAL_2 is determined, everyone can interpolate "in the exponent" and compute $U_T = \sum_{i \in \mathsf{QUAL}'} \lambda_i \cdot U_{T,i} = s \cdot E$, where the λ_i's are the Lagrange interpolation coefficients. The group element U_T is the next output of the beacon. Note that the adversary has no influence over the U_T's, they are always set as $U_T = s \cdot \mathsf{Hash}'(T)$. On the other hand, before the shares $U_{T,i}$ are broadcast, the value U_T is unpredictable (indeed pseudorandom) from the adversary's point of view.

2.9 More Optimizations

While quite efficient as-is, in many settings there are additional optimizations that can significantly improve the performance of our protocols, such as committing to evaluation points (rather than coefficients) and using optimistic parameters with safe fallback when sub-sampling committees (See Appendix A). Also, in Appendix H we discuss the dishonest majority case for a mixed malicious/semi-honest adversary model.

2.10 Parameters and Performance

Various parameters and performance analysis are provided in Appendix C, here we give a very brief overview.

[14] More precisely, there is a public commitment $\hat{\mathbf{F}}$ of $\mathbf{F}$ from which anyone can derive a Feldman commitment $\sigma_i \cdot G$ of σ_i. See Section 2.1.

To get enough honest parties in HOLD, we need to have $n \geq 3t + 2a - 1$, and we often assume that this holds with equality. Then we set $d_1 = |\mathsf{QUAL}| = n - t$ and get $b = n - 2t$, hence we can get as many as $ab = a(n - 2t)$ signatures for each run of the protocol. Some example numbers are $n = 10, t = 2, a = 2, b = 6$ (12 signatures per run), or $n = 64, t = 15, a = 10, b = 34$ (340 signatures per run). In the setting of a large open system where committees are sub-sampled, we can even sign more messages in each run without reducing resiliency: for example, assuming 80% honest parties, we can sub-sample a committee of size $n = 992$ with $t = 336, a = 40, b = 320$, and sign 12880 messages in each run.[15]

If we set $t = a = n/5$, we can sign $3n^2/25$ messages per run, with an amortized bandwidth of fewer than 35 scalars/group elements broadcasted per signature. For the sub-sampling parameters above with $n = 992$ (with a group of size $\approx 2^{256}$), the total broadcast bandwidth is only under 100MB.

Given parameters n, t, a, the parties broadcast less than $4n^2$ scalars and group elements (in total). If we set $t = a = n/5$, we can sign $3n^2/25$ messages per run, with an amortized bandwidth of fewer than 35 scalars/group elements broadcasted per signature. For the sub-sampling parameters above with $n = 992$ (with a group of size $\approx 2^{256}$), the total broadcast bandwidth is only under 100MB.

In terms of computation, the most expensive part is multiplying the super-invertible matrix in the exponent (which is needed to compute the public R's). This part takes at most $at(n - 2t)$ products (using a naive algorithm), which is t scalar-elements multiplications per signature. But as we explain in Section 2.4, we can use much more efficient matrix-multiplication to reduce it, or just use small scalars. Even without these optimizations, for the example with $n = 992$ every party needs to compute about five million such scalar-element multiplications, which can be done within 5 minutes with a single thread. With the small-scalar Vandermonde optimization from above, this goes down to about 1 minute.

Since the super-invertible matrix multiplication is the most expensive part of the protocol, we wrote code to benchmark actual performances for both our possible optimizations from Section 2.4. Table 1 in Appendix B.3 on Page 32 shows the results for various values of b, when $b = t$ is a power of two. For $b = t = 256$, our first solution provides a 29× speed up compared to the naive solution and only takes 682ms when $a = 1$ (on a single core of a 2.20GHZ AMD EPIC 7601 CPU). Even with $a = 64$, the total super-invertible matrix multiplication would take less than 1 minute on a single-core. In addition, this operation is trivially parallelizable, computations for each of the a packed values are completely independent of each other and can be run on different threads.

For $b = t = 512$, our second solution becomes faster and provides a 28× speed up compared to the naive solution. It only takes 2.80s to compute the super-invertible matrix multiplication in that setting for $a = 1$, on a single core.

Parameters: Integers $n, t, a \geq 1, d = t + a - 1, d' = t + 2a - 2$.

Setup: (Parties: $P_1, \ldots, P_n$)

- Each P_i holds a share $\sigma_i = \mathbf{F}(i)$, where $\mathbf{F}$ is a random degree-d polynomial subject to $\mathbf{F}(0) = \mathbf{F}(-1) = \ldots = \mathbf{F}(-a+1)$. Denote $s = \mathbf{F}(0)$.
- Public keys $S = s \cdot G$ and $S_i = \sigma_i \cdot G$ are publicly known.

Ephemeral randomness generation

1. Each $P_i, i \in [n]$, chooses a random degree-d' polynomial $\mathbf{H}_i$; it broadcasts Feldman commitments to $\mathbf{H}_i$ of the form $\hat{\mathbf{H}}_i(v) = \mathbf{H}_i(v) \cdot G$ for $v \in [-a+1, t+a-1]$. Encrypt the share $\rho_{ij} = \mathbf{H}_i(j)$ under the public key of P_j for all $j \in [n]$, and boardcast all the resulting ciphertexts.
2. $P_1, \ldots, P_n$ run the protocol from Fig. 3 to agree on $\mathsf{QUAL}, \mathsf{BAD}, \mathsf{HOLD} \subseteq \{P_1, \ldots, P_n\}$ with $d_0 = |\mathsf{HOLD}| = n - t$, $d_1 = |\mathsf{QUAL}| + |\mathsf{BAD}| = n - t$, and every $P_j \in \mathsf{HOLD}$ holds valid shares from all the dealers in QUAL.
3. Set $b = |\mathsf{QUAL}| - (t - |\mathsf{BAD}|)$; $\Psi = [\psi_i^u] \in \mathbb{Z}_p^{b \times |\mathsf{QUAL}|}$ a super-invertible matrix.
   For $u \in [b]$, $v \in [a]$, define $\mathbf{H}^u(\cdot) = \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i(\cdot)$, $r^{u,v} = \mathbf{H}^u(1-v)$, $R^{u,v} = r^{u,v} \cdot G$.[a]
   Each $P_j \in \mathsf{HOLD}$ sets $\rho_j^u = \mathbf{H}^u(j) = \sum_{i \in \mathsf{QUAL}} \psi_i^u \rho_{ij}$ for all $u \in [b]$.

Signature share generation On input messages $M^{u,v}, u \in [b], v \in [a]$:

Each $P_j \in \mathsf{HOLD}$ sets $\delta = \mathsf{Hash}\big(S, \mathsf{QUAL}, \{(R^{u,v}, M^{u,v}) : u \in [b], v \in [a]\}\big)$ and $\Delta = \delta \cdot G$.

Then, it runs the following procedure, in parallel, for each $u \in [b]$:

1. Computes $e^{u,v} = \mathsf{Hash}(S, \Delta + R^{u,v}, M^{u,v})$ for $v \in [a]$;
2. Computes the degree-$(a-1)$ polynomial $\mathbf{Z}^u$, with $\mathbf{Z}^u(1-v) = e^{u,v}$ for $v \in [a]$.
3. Outputs *signature share*: $\pi_j^u = \mathbf{Z}^u(j) \cdot \sigma_j + \rho_j^u$.
   Note: $\pi_j^u = \mathbf{Y}^u(j)$ for the degree-d' polynomial $\mathbf{Y}^u = \mathbf{Z}^u \cdot \mathbf{F} + \mathbf{H}^u$

Schnorr signature assembly (from signature shares)
For each issued signature share π_j^u verify, using commitments to $\mathbf{H}_i, i \in \mathsf{QUAL}$, and public key $S_j = \mathbf{F}(j) \cdot G$, that $\pi_j^u \cdot G = \mathbf{Z}^u(j) \cdot S_j + \mathbf{H}^u(j) \cdot G$.
When collecting $d'+1$ verified shares π_j^u, reconstruct the polynomial $\mathbf{Y}^u$ and for all $v \in [a]$ set $\phi^{u,v} = \mathbf{Y}^u(1-v)$. (Note: $\phi^{u,v} = \mathbf{Y}^u(1-v) = \mathbf{Z}^u(1-v) \cdot \mathbf{F}(1-v) + \mathbf{H}^u(v) = e^{u,v} \cdot s + r^{u,v}$.)
For $v \in [a], u \in [b]$, output the Schnorr signatures $(\Delta + R^{u,v}, \delta + \phi^{u,v})$ on message $M^{u,v}$.

[a] The values $R^{u,v}$ can be computed from commitments to the polynomials $\mathbf{H}_i$, hence public information.

Figure 1: SPRINT Scheme in the Static-Committee Setting

3 The SPRINT Protocols

3.1 Static-Committee Setting

We begin with our base protocol shown in Fig. 1, namely, a robust threshold Schnorr signature scheme for the static-committee case where the set of parties is fixed. It follows the design and rationale presented in Section 2 (particularly, till Section 2.6), resulting in a two-round ephemeral randomness generation phase (dependent on the number of messages to be signed but not on the messages themselves) followed by a *non-interactive* signing procedure. It considers n parties of which at most t are corrupted, and is given a packing parameter a and an amplification (via a super-invertible matrix) parameter b. It assumes an asynchronous broadcast channel. The protocol consists of three parts. An initial setup stage where parties obtain shares σ_i of a long-term secret key s, and corresponding public key $S = s \cdot G$, and $S_i = \sigma_i \cdot G$ are made public. We assume that sharing the secret key uses packed secret sharing, namely, the parties' shares σ_i lie on a polynomial $\mathbf{F}$ of degree $d = t + a - 1$, such that $\mathbf{F}(0) = \mathbf{F}(-1) = \ldots = \mathbf{F}(-a+1) = s$. This initial setup can be done via a distributed key generation (DKG) protocol or another secure procedure.

The second part is the *generation of ephemeral randomness for Schnorr signatures.* Following the DKG blueprint of [36, 17], each party P_i shares a random polynomial $\mathbf{H}_i$ by transmitting the value $\mathbf{H}_i(j)$ to each other party P_j and committing to $\mathbf{H}_i(\cdot)$ over a public broadcast channel. Our application allows for the use of the more efficient Feldman secret sharing [13]. In our case, parties commit to their polynomials $\mathbf{H}$ by broadcasting values $\mathbf{H}(v) \cdot G$ for $d' + 1$ different evaluation points v where d' is the degree of $\mathbf{H}$ (specifically, in our case, this set is defined as the interval $[-a+1, t+a-1]$).

A central part of such a protocol is for the parties to agree on sets of dealers (denoted $\mathsf{QUAL}, \mathsf{BAD}$) that shared their polynomials correctly/badly, and a large enough set of parties (denoted HOLD) that received correct sharings from all parties in QUAL. In Section 4.1 we describe an implementation of such a protocol over an asynchronous atomic broadcast channel.

The source of efficiency for SPRINT is the use of packing to share a secrets at a little more cost than sharing just one and attaining further amplification, by a factor of b, using super-invertible matrices [26] (see Section 2.4). Here, b is the number of rows in the super-invertible matrix Ψ, e.g., a Vandermonde matrix, and is set to its largest possible value (as analysis shows), $b = |\mathsf{QUAL}| - (t - |\mathsf{BAD}|)$. (Smaller values of b can be used too, if fewer messages need to be signed.) Once the randomness generation procedure is completed, each party in HOLD generates (non-interactively) signature shares consisting of a point on a polynomial $\mathbf{Y}$ that when reconstructed (via interpolation of $d' + 1$ signature shares) can be evaluated on a points to achieve a signatures. Remarkably, using super-invertible matrices one can generate b different polynomials $\mathbf{Y}$, hence resulting in $a \cdot b$ signatures at the cost of a single execution of the (interactive) randomness generation procedure.

In all, we have that after the randomness generation procedure, parties generate their shares of the signatures without any further interaction. Each party P_j computes locally their signature shares $\pi_j^u, u \in [b]$ and publishes them. Reconstructing the signature for each batch of a messages $M^{u1}, \ldots M^{ua}$ can be done by interpolation from any $d' + 1$ correct signature shares π_j^u. Moreover, signature shares can be verified individually by a Schnorr-like validation $\pi_j^u \cdot G = \mathbf{Z}^u(j) \cdot S_j + \rho_j^u \cdot G$,

[15] We need $n \geq 657$ to get (statistical) safety failure $< 2^{-80}$ (and liveness failure $< 2^{-11}$), without packing (i.e., $a = 1$). Setting $a = 40$ only requires $n \geq 992$ while multiplying the number of messages that can be signed by 40 and while providing the same safety guarantees. This is because we have less than $(n-1)/3$ corrupted parties selected in each committee with overwhelming probability. See details in Appendix D.1.

where all the required information is public. Thus, invalid signature shares can be discarded.

An additional ingredient in the protocol is the use of the "mitigation value" $\delta = \mathsf{Hash}(S, \mathsf{QUAL}, \{(R^{u,v}, M^{u,v}) : u \in [b], v \in [a]\})$ needed to achieve security when running the $a \cdot b$ signatures in parallel, as explained in Section 2.3.

Security of the SPRINT protocol from Fig. 1 is proven in Theorem 5.

3.2 The Dynamic/Proactive Setting

The adaptation of SPRINT to the dynamic setting is shown in Fig. 2. See also Section 2.7. It requires two types of sharings. One is ephemeral randomness generation as in the static setting, where dealers have no input, and they just share random polynomials. The other is a share refresh (i.e., proactive resharing), in which the dealers have shares of the long-term secret, and they refresh the sharing of that secret to the shareholders. These two sharings are enabled by (almost) the same DKG-like protocol, both using the agreement protocol from Fig. 5 with the same set HOLD and two QUAL sets for the two sharings. Note that the use of the same set HOLD for both sharings is crucial to guarantee that enough parties (those in HOLD) have *both* shares of the secret s and of the ephemeral randomness as needed for generating signatures. Proving security of this protocol is very similar to the static case; see more details in Appendix G.7.

A note on the "traditional" proactive setting. The proactive setting [33, 24, 23] was originally envisioned as a periodic operation, say every week, in order to heal the system from active and passive corruptions. When running SPRINT in such a scenario, one would not want to perform a share refresh with each run of the signature generation protocol (Fig. 1) but only at the end of a full proactive period. However, by decoupling the two sharings (refresh and randomness generation), we lose the ability to use the same set HOLD for both cases. This raises a liveness issue: If the share refresh ends with a set HOLD of size $n - t$ and a subsequent execution of SPRINT ends with a *different* set HOLD' of the same size, then it may be the case that the intersection of these two sets will have less than $t + 1$ uncorrupted parties, hence unable to create signatures.

However, the traditional proactive setting already assumes the share refresh to happen within a more controlled environment.[16] Thus, it makes sense to consider a more synchronous setting (with monitored and resolved delays) during refresh in which case the share refresh operation can be assumed to be completed after a defined amount of time for non-adversarial servers. In this case, parties that did not make it to HOLD by that time will be disqualified from participating in signature generation until the next proactive execution and be counted towards the bound t on corrupted parties. This guarantees that all honest parties in sets HOLD created by runs of SPRINT until the next refresh period will have valid shares of the secret s.

4 The Agreement Protocol

For agreement, we observe that in the static setting we can have a more efficient agreement protocol than in the proactive/dynamic setting. As a result, we present two protocols of a very similar flavor for the task of reaching agreement. In this section we describe in detail the base agreement protocol, which achieves the best results for the static setting. Then we sketch the enhancements that we

[16] E.g., it assumes human intervention to replace or reboot servers, to export public keys from new servers or servers that choose new (encryption) keys, etc. (see [24]).

Parameters: Integers $n, t, a \geq 1, d = t + a - 1, d' = t + 2a - 2$.

Parties: Dealers $D_1, \ldots, D_n$, shareholders $P_1, \ldots, P_n$

Setup: (D_i's)

- Each D_i holds a share $\sigma_i = \mathbf{F}(i)$, where $\mathbf{F}$ is a random degree-d polynomial subject to $\mathbf{F}(0) = \mathbf{F}(-1) = \ldots = \mathbf{F}(-a+1)$. Denote $s = \mathbf{F}(0)$.
- Public keys S and $S_i = \sigma_i \cdot G$ are publicly known.

Ephemeral randomness generation and Re-sharing (The D_i's and P_j's)

1. Each $D_i, i \in [n]$, with share $\sigma_i = \mathbf{F}(i)$ chooses:
   - A random degree-d' polynomial $\mathbf{H}_i$;
   - A degree-d polynomial $\mathbf{F}_i$, random subject to $\mathbf{F}_i(0) = \cdots = \mathbf{F}_i(1-a) = \sigma_i$.

   D_i broadcasts Feldman commitments to $\mathbf{F}_i, \mathbf{H}_i$;

   D_i encrypts $\rho_{ij} = \mathbf{H}_i(j)$ and $\sigma_{ij} = \mathbf{F}_i(j)$ under P_j's key $\forall j \in [n]$, and broadcasts all these ciphertexts.

2. $P_1, \ldots, P_n$ run the protocol from Fig. 5 to agree on $\mathsf{HOLD} \subseteq \{P_1, \ldots, P_n\}$, $\mathsf{QUAL}_1, \mathsf{QUAL}_2 \subseteq \{D_1, \ldots, D_n\}$ with $d_0 = |\mathsf{HOLD}| = n - t$, $d_1 = |\mathsf{QUAL}_1| = n - t$, $d_2 = |\mathsf{QUAL}_2| = t + a$, where every $P_j \in \mathsf{HOLD}$ received valid shares ρ_{ij} from all the dealers in QUAL_1 and valid shares σ_{ij} from all the dealers in QUAL_2.[a]

3. Set $b = |\mathsf{QUAL}_1| - t$; $\Psi = [\psi_i^u] \in \mathbb{Z}_p^{b \times |\mathsf{QUAL}_1|}$ a super-invertible matrix.

   For $u \in [b]$, $v \in [a]$, define $\mathbf{H}^u(\cdot) = \sum_{i \in \mathsf{QUAL}_1} \psi_i^u \mathbf{H}_i(\cdot)$, $r^{u,v} = \mathbf{H}^u(1-v)$, $R^{u,v} = r^{u,v} \cdot G$.

   Each $P_j \in \mathsf{HOLD}$ sets $\rho_j^u = \sum_{i \in \mathsf{QUAL}_1} \psi_i^u \rho_{ij}$ for all $u \in [b]$.

4. Each $P_j \in \mathsf{HOLD}$ sets $\sigma'_j = \sum_{i \in \mathsf{QUAL}_2} \lambda_i \mathbf{F}_i(j)$, the λ_i's are the Lagrange coefficients for QUAL_2.

   Let $\mathbf{F}' = \sum_{i \in \mathsf{QUAL}_2} \lambda_i \mathbf{F}_i$; a commitment to $\mathbf{F}'$ is obtained from those of the $\mathbf{F}'_i$'s.

Signature generation and assembly Same as in the static case in Fig. 1 but using polynomial $\mathbf{F}'$ instead of $\mathbf{F}$ in that figure.

[a] Valid σ_{ij} mean in particular that $\mathbf{F}'_i$ indeed has the required format, with $\mathbf{F}_i(0) = \cdots = \mathbf{F}_i(1-a) = \sigma_i = F(i)$.

Figure 2: SPRINT Scheme in the Dynamic-Committee Setting

need for the dynamic/proactive setting in what we refer to as the full protocol, which is described in Appendix E.

This protocol is designed to work over an asynchronous total-order (aka atomic) broadcast channel. Recall that a total-order broadcast channel provides the following guarantees:

- *Eventual delivery.* A message broadcasted by an honest party will eventually be seen (unmodified) by all honest parties. However, the adversary can change the order in which messages are delivered to the broadcast channel.
- *Prefix consistency.* Considering the views of the broadcast channel at a given time by two different honest parties, the view of one is a prefix of the other.
- *Authenticity.* Messages that are received on behalf of honest parties were indeed sent by those honest parties.

We also assume a PKI, i.e., each party has an encryption public key that is known to all other parties. The protocol below uses only the broadcast channel for communication, private messages are sent by encrypting them and broadcasting the ciphertext.

Time and Steps. While a total-order broadcast channel is not synchronous, and thus it has no absolute notion of time, we are still ensured that the parties all see the same messages in the same order. We can therefore define a "step T" as the time when the T'th message is delivered. Even though different parties may see it at different times, they will all agree on the message that was delivered at step T. If we have a protocol action that is based only on the messages that appeared on the broadcast channel up to (and including) the T'th message, we are ensured that all the honest parties will take the same action, and they will all know that they did it at "step T".

In the description below we distinguish between dealers and shareholders. The protocol begins with the dealers broadcasting messages, then the shareholders engage in a protocol among themselves based on the dealer messages that they see on the channel. For every dealer message and every shareholder, the shareholder either accepts this message or complains about it.

An important technique in our protocol is the use of "verifiable complaints": This is a complaint by a shareholder about a dealer, that will be accepted by all other honest shareholders. (In our context, it will be implemented by proving that the message sent by that dealer is invalid.) We say that a dealer message is "locally bad" for shareholder P_j, if that shareholder is able to generate a verifiable complaint against it. Importantly, we assume that it is impossible to produce a verifiable complaint against messages sent by honest dealers.

We denote the number of dealers as n_1, at least d_1 of them are assumed to be honest. The protocol is run among a set of n_0 shareholders, at least d_0 of which are assumed to be honest. We require that this base protocol terminates, and that all honest shareholders output the same sets HOLD, $\mathsf{QUAL}, \mathsf{BAD}$, where HOLD is a subset of the shareholder set with $|\mathsf{HOLD}| \geq d_0$, and $\mathsf{QUAL}, \mathsf{BAD}$ are disjoint subsets of the dealer set with $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$.

The base protocol is described in Fig. 3 and proven in Theorem 1. Here each shareholder initially sets QUAL to the first d_1 dealers whose broadcast message they receive. Then each shareholder broadcasts a message specifying which of these d_1 dealers sent correct shares and complaining about the ones that did not. Thereafter, each shareholder continuously adds to HOLD the shareholders whose message appeared on the channel, and moves dealers from QUAL to BAD when they see a verifiable complaint against them on the channel. The protocol terminates once HOLD reaches size d_0.

Theorem 1. Consider an execution of the base agreement protocol from Fig. 3 over a total-order broadcast channel, among a set of n_0 shareholders of which at least d_0 are honest. Assume that

Parameters: n_0, d_0, n_1, d_1 (should agree on $|\mathsf{HOLD}| \geq d_0$, $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$).

Precondition: We have up to n_1 dealers, at least d_1 of which are honest. We also have n_0 shareholders, at least d_0 of them are honest.

Shareholder P_j:

Initialize $\mathsf{HOLD} = \mathsf{QUAL} = \mathsf{BAD} = \emptyset$

1. **Enlarge QUAL.** While $|\mathsf{QUAL}| < d_1$, when receiving the (first) broadcast message of the right format[a] from dealer D_i, set $\mathsf{QUAL} := \mathsf{QUAL} \cup \{D_i\}$.

2. **Broadcast Complaints.** Once $|\mathsf{QUAL}| \geq d_1$, broadcast all the verifiable complaints against dealers in QUAL whose message was locally bad, in a single broadcast message. If this set is empty, broadcast the empty set.

3. **Contract and fix QUAL, BAD.** Collect all the valid complaint-sets (i.e., the ones whose complaints can be verified, or the empty set). Once there are d_0 valid complaint-sets, set $\mathsf{QUAL} := \mathsf{QUAL} \setminus \{D_i\}$ and $\mathsf{BAD} := \mathsf{BAD} \cup \{D_i\}$ for each verifiable complaint against dealer D_i;

4. **Fix HOLD.** To the first d_0 shareholders who broadcasted a valid complaint-set.

[a] In our context, a message has the right format if it contains all the commitments and ciphertexts that it was supposed to have.

Figure 3: Base protocol for agreeing on QUAL, BAD, HOLD

at most n_1 dealers broadcast messages, at least d_1 of these dealers are honest, and no verifiable complaint can be constructed against any honest dealer. Then all honest shareholders will eventually terminate, all outputting the same sets with $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$. Moreover:

- No shareholder in HOLD complained against any dealer in QUAL; and
- Every dealer in BAD has at least one shareholder in HOLD that lodged a verifiable complaint against them.

The proof is in Appendix F.

4.1 Agreement in SPRINT, the Static Case

To instantiate the base agreement protocol in SPRINT, we need to set the parameters n_0, d_0, n_1, d_1 and specify how the dealer's messages and verifiable complaints are generated and verified.

In our protocols, a dealer's message is just a Shamir sharing of secrets via polynomials. In the static case, we have one pair of QUAL, BAD for the DKG polynomials. We assume a PKI, and the dealers encrypt and broadcast all the shares under the public keys of their intended recipient, and also broadcast Feldman commitments to the polynomials themselves.

There are checks that all shareholders can perform on public information that the dealers broadcast, i.e. verifying that the committed polynomials are of the right degree, and that the dealer's message includes all the ciphertexts that it is supposed to. However, each shareholder is the only one who can check if the share encrypted under their public key is consistent with the committed polynomial.

Dealer D_i (sharing a degreed-d polynomial, $\mathbf{F}_i(X)$):

1. Compute $\hat{\mathbf{F}}_i = \{\mathbf{F}_i(k) \cdot G : k \in [-a+1, \ldots, d-a]\}$;
2. Let $\sigma_{ij} = \mathbf{F}_i(j)$ and $E_{ij} = ENC_{PK_j}(\sigma_{ij})$ for all $j \in [n]$;
3. Broadcast $\left(\{E_{i1}, \ldots, E_{in}\}, \hat{\mathbf{F}}_i\right)$.

Shareholder P_j**:**

1. Decrypt $\sigma_{ij} = DEC_{SK_j}(E_{ij})$ and verify $\sigma_{ij} \cdot G \stackrel{?}{=} \sum_{k=1-a}^{d-a} \lambda_{i,j,k} \cdot (\mathbf{F}_i(k) \cdot G)$, with $\lambda_{i,j,k}$ the relevant Lagrange coefficients;
2. If verification failed, create a verifiable complaint against E_{ij}, consisting of the decrypted value σ_{ij} and a proof-of-correct-decryption of E_{ij} relative to PK_j.

Figure 4: Dealer messages and shareholder complaints

If the encrypted share is *not* consistent with the committed polynomial, the shareholder will create a verifiable complaint, using the fact that the dealer's message is visible to all. A verifiable complaint from shareholder P_j, denoted π_{ji}, consists of the decrypted value from the ciphertext that D_i sent to P_j, and a proof-of-correct-decryption relative to P_j's public key.[17] Once other parties see the decrypted value they can all verify that the share indeed is not consistent with the committed polynomial.

The dealer messages and shareholder complaints are described in Fig. 4.

Parameters in the static-committee setting. In the static-committee setting, each dealer shares a single random polynomial H_i of degree $d' = t+2a-2$. To ensure that the resulting random polynomials can be recovered we need at least $d'+1$ honest parties in HOLD, so we have to set $d_0 \geq t+d'+1 = 2t+2a-1$. But we can set it even bigger, it can be as large as $n-t$ since we know that there are at least as many honest shareholders. (This implies that we need $n-t \geq 2t+2s-1$, namely $n \geq 3t+2a-1$.)

We note that for the DKG protocol, the size of QUAL is unrelated to the degree of the polynomials H_i. The only constraint on it is that to get b output random polynomials we need $|\mathsf{QUAL}| + |\mathsf{BAD}| = d_1 \geq b+t$. To get the best amortized cost, we want to make b as large as possible, which means using as large an initial set $\mathsf{QUAL} \cup \mathsf{BAD}$ as we can get. Every party can serve as a dealer for the DKG protocol, so we have at least $n-t$ honest dealers and can set $d_1 = n-t$ (and therefore $b = n-2t$).

Hence, we run the agreement protocol with parameters $d_0 = d_1 = n-t$. (If we have fewer messages to sign, we can do with a smaller b, which means smaller d_1, any value $d_1 > t$ would work.)

4.2 Agreement in the Dynamic/Proactive setting

In this setting, dealers share two types of polynomials, random polynomials $\mathbf{H}_i$ of degree $d' = t+2a-2$ for the DKG, and packed re-sharing polynomials $\mathbf{F}_i$ of degree $d = t+a-1$.

[17] The proof-of-decryption can be very simple: a proof of equality of discrete logs if using ElGamal encryption for the shares, or showing an inverted RSA ciphertext if using RSA-based encryption.

Here we must rely on stronger agreement guarantees. For the static case, it was enough to ensure that in a setting with d_1 honest dealers, we will end up with $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$, this was enough to ensure $d_1 - t$ honest dealers in QUAL (which is the best we can so in the worst case, and is what's needed for the DKG). Now, however, we need to ensure the stronger condition $|\mathsf{QUAL}| \geq d_1$, since this is what's needed for re-sharing the secret.

We therefore augment the agreement by running multiple iterations of the base protocol. In every iteration, we enlarge QUAL until it reaches side d_1, then have one round of complaints and potentially move some more dealers from QUAL to BAD. This is repeated until no more dealers are added to BAD, at which point we have $|\mathsf{QUAL}| \geq d_1$. (Note that at the beginning of each iteration, we always have enough honest dealers whose messages were not yet incorporated in the protocol to reach QUAL of size-d_1 in this iteration.)

Another enhancement to the protocol is that we now have two separate QUAL's (and corresponding two BAD's): one pair $\mathsf{QUAL}_1, \mathsf{BAD}_1$ for the H_i's, and another pair $\mathsf{QUAL}_2, \mathsf{BAD}_2$ for the F_i's. We however only have one shareholder set HOLD (since we need the same shareholders to get both a share of the key and a share of the ephemeral secrets). The protocol is in Appendix E.

Parameters in the dynamic-committee setting. Here we have parameters n_0, d_0 for HOLD and n_1, d_1 for $\mathsf{QUAL}, \mathsf{BAD}$ as before (for the H_i's), but in addition also n_2, d_2 for $\mathsf{QUAL}', \mathsf{BAD}'$. For the $\mathbf{H}_i$'s we have the same parameters as above, $n_0 = n_1 = n$ and $d_0 = d_1 = n - t$. For the $\mathbf{F}_i$'s, we need $d + 1 = t + a$ dealers in QUAL' in order for shareholders in HOLD to be able to recover their shares, so we set $d_2 = t + a$.

All the dealers in QUAL' must have shares of the long-term secret, so they had to be in HOLD in the previous epoch. Hence, the pool of dealers could be as small as $n_2 = d_0 = n - t$, and t of them could be corrupted, so we cannot set d_2 any larger than $n - 2t$. This implies the constraint $d_2 = n - 2t \geq t + a$ or $n \geq 3t + a$. This constraint is weaker than the constraint $n \geq 3t + 2a - 1$ from above.

Acknowledgements. We thank Victor Shoup for mentioning to us the solution using a Vandermonde matrix for fast multiplication by a super-invertible matrix.

References

[1] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO*, 2023.

[2] arkworks contributors. arkworks zksnark ecosystem. https://github.com/arkworks-rs/.

[3] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer, Heidelberg, March 2008.

[4] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th ACM STOC*, pages 52–61. ACM Press, May 1993.

[5] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic curve fast fourier transform (ECFFT) part I: low-degree extension in time $O(n \log n)$ over all finite fields. In

Nikhil Bansal and Viswanath Nagarajan, editors, *SODA 2023, Florence, Italy, January 22-25, 2023*, pages 700–737. SIAM, 2023.

[6] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. *J. Cryptol.*, 35(4):25, 2022.

[7] William Borgeaud. ECFFT algorithms on the BN254 base field, 2023.

[8] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 2019.

[9] Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. How to prove Schnorr assuming Schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, 2021. https://eprint.iacr.org/2021/1375.

[10] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. Cryptology ePrint Archive, Report 2022/1389, 2022. https://eprint.iacr.org/2022/1389.

[11] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew K. Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy*, pages 2518–2534. IEEE Computer Society Press, May 2022.

[12] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1084–1101, 2019.

[13] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*, pages 427–437. IEEE Computer Society Press, October 1987.

[14] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710. ACM Press, May 1992.

[15] Chaya Ganesh and Arpita Patra. Optimal extension protocols for byzantine broadcast and agreement. *Distributed Comput.*, 34(1):59–77, 2021.

[16] François Garillot, Yashvanth Kondi, Payman Mohassel, and Valeria Nikolaenko. Threshold Schnorr with stateless deterministic signing from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 127–156, Virtual Event, August 2021. Springer, Heidelberg.

[17] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

[18] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *17th ACM PODC*, pages 101–111. ACM, June / July 1998.

[19] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO*, 2022.

[20] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Report 2022/506, 2022. https://eprint.iacr.org/2022/506.

[21] Jens Groth and Victor Shoup. On the security of ECDSA with additive key derivation and presignatures. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 365–396. Springer, Heidelberg, May / June 2022.

[22] Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. Cryptology ePrint Archive, 2023. https://eprint.iacr.org/2023/1175.

[23] Amir Herzberg, Markus Jakobsson, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In Richard Graveman, Philippe A. Janson, Clifford Neuman, and Li Gong, editors, *ACM CCS 97*, pages 100–110. ACM Press, April 1997.

[24] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 339–352. Springer, Heidelberg, August 1995.

[25] Martin Hirt, Christoph Lucas, and Ueli Maurer. A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 203–219. Springer, Heidelberg, August 2013.

[26] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 463–482. Springer, Heidelberg, August 2006.

[27] Snehil Joshi, Durgesh Pandey, and Kannan Srinathan. Atssia: Asynchronous truly-threshold schnorr signing for inconsistent availability. In Jong Hwan Park and Seung-Hyun Seo, editors, *Information Security and Cryptology – ICISC 2021*, pages 71–91, Cham, 2022. Springer International Publishing.

[28] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[29] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Heidelberg, October 2020.

[30] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374, 2022. https://eprint.iacr.org/2022/374.

[31] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*,

volume 179 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[32] Wafa Neji, Kaouther Blibech, and Narjes Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Security and Communication Networks*, 9(17):4585–4595, 2016.

[33] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *10th ACM PODC*, pages 51–59. ACM, August 1991.

[34] Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Efficient statistical asynchronous verifiable secret sharing with optimal resilience. In Kaoru Kurosawa, editor, *ICITS 09*, volume 5973 of *LNCS*, pages 74–92. Springer, Heidelberg, December 2010.

[35] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 522–526. Springer, Heidelberg, April 1991.

[36] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[37] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT'96*, volume 1163 of *LNCS*, pages 252–265. Springer, Heidelberg, November 1996.

[38] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2551–2564. ACM Press, November 2022.

[39] Victor Shoup. The many faces of Schnorr. Cryptology ePrint Archive, 2023. https://eprint.iacr.org/2023/1019.

[40] Douglas R. Stinson and Reto Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In Vijay Varadharajan and Yi Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 417–434. Springer, Heidelberg, July 2001.

[41] Pieter Wuille. BIP 0032, Bitcoin improvement proposal. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki, 2012.

[42] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew K. Miller. hbACSS: How to robustly share many secrets. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*, 2022.

A More Optimizations

While quite efficient as-is, in many setting there are additional optimizations that can significantly improve the performance. A few are described below.

Committing to evaluation points. Clearly, from a security perspective, there is no difference if the commitment to the polynomials $\mathbf{H}_i, \mathbf{F}_i$ is done by committing to their coefficients or to their values at some points (or any mix therefore). But different choices have an effect on the computational complexity of the protocols. In particular, our protocol embeds the various secrets at evaluation points $0, -1, -2, \ldots, -a+1$, so it makes sense to commit to these values instead of the coefficients, this will allow everyone to verify these values without having to compute them every time.

We use the convention that a commitment to a degree-$(d+a)$ polynomial F consists of the $d+a+1$ group elements $F(v) \cdot G$ for all $v \in [-a+1, d+1]$. This way, we directly get $s^v \cdot G$ for all the a embedded secrets s^v, and also $\sigma_i \cdot G$ for the shares σ_i of the first $d+1$ parties. For the shares of the other $n-d-1$ parties, $i > d+1$, obtaining $\sigma_i \cdot G$ requires computing "linear combinations in the exponent" using the appropriate Lagrange coefficients.

Also, in some cases, we know that some embedded secrets are equal, so we only need to commit to them once. In particular, our share-refresh protocol uses polynomials F_i such that $F_i(-a+1) = \cdots = F_i(0) = \sigma_i = F(i)$. Since $\sigma_i \cdot G$ is known ahead of time, there is no need for D_i to send that group element again. Hence, even though F_i is a polynomial of degree $t+a-1$, the dealer D_i only needs to send t group elements to commit to it.

Optimistic parameters with safe fallbacks. When sub-sampling the committees from a large population, we need them to be large enough to ensure both safety and liveness with high probability. However, we may set different confidence levels for safety and liveness. In particular, we require that safety holds except with a negligible probability (statistical security of $1-2^{-80}$), but allow liveness to be violated with small but not quite non-negligible probability (99.95% of progress), and re-run the protocol with a new committee in the very rare cases that it failed to make progress. Moreover, it may make sense to start from a relatively weak liveness guarantee, in the hope that the protocol is completed, and only switch to larger committees and stronger liveness guarantee if the initial optimistic attempt fails to produce signatures.

For example, our "main parameters" (see Appendices C.5 and J.1) are chosen to ensure (statistical) 2^{-80} assurance for safety and 2^{-11} assurance for liveness when 80% of the population are honest. But we can start from a smaller-sized committee that still gives 2^{-80} assurance for safety at 80% honest, but only (say) 2^{-8} liveness assurance at 95% honest. Namely, it gives the same assurance as before against the adversary learning the secret, but even an adversary controlling only 5% of the parties already has about $1/300$ chance of preventing the generation of signatures. The parties can run the protocol with these parameters for a while, and if the signatures are not generated for a while, then timeout and fall back on the more conservative parameters above.

Importantly, in the proactive case, this optimistic approach still ensures the following weak liveness guarantee: the adversary cannot push the system into an unrecoverable state where honest parties do not have enough shares to reconstruct the long-term secret key. Concretely, if the protocol executed with the optimistic parameters succeeds, then at least $d+1$ honest shareholders managed to reconstruct the new shares of the long-term secret key.

Signing more messages. Our QUAL-agreement protocol for DKG ensures at least $n-t$ qualified dealers, but in "normal operation" we can expect more dealers to behave honestly, perhaps even all of them. In practical deployments we can modify our QUAL-agreement protocol, letting

shareholders wait a little for more dealers to send shares, before broadcasting their support messages. (For example, wait a few more blocks if we use a blockchain as our broadcast channel.) This way, if many more dealers are honest and synchronized, we can end up with a larger QUAL set. This in turn will let us sign $a \cdot (|\mathsf{QUAL}| - t) > a \cdot b$ messages in this run. In the best case, we could have as many as $|\mathsf{QUAL}| = n$, letting us sign $a(n-t)$ messages.

User aggregation. In settings such as proof-of-stake blockchains, we may view each token as a party for the purpose of our protocols, and therefore a node holding many tokens will have to play the role of many parties. In that case, we can generate, verify, and use the shares of all these parties together. In particular, a party controlling multiple evaluation points can save on the following actions:

- When playing a dealer and sending the encrypted shares, it is possible to use hybrid encryption, establishing just a single shared key (per recipient) and using that key to send the shares corresponding to all the evaluation points.
- When playing a shareholder and verifying the shares corresponding to multiple evaluation points (say ℓ points), the shareholder needs to check equality of the form $\vec{u} \cdot G = \Gamma \cdot \vec{V}$, where $\vec{u}, \Gamma$ are vector/matrix over $\mathbb{Z}_p$ and $\vec{V}$ is a vector of group elements. Namely, $\vec{V} \in \mathbb{G}^{d'+1}$ consists of the dealer's commitment to their polynomial, $\Gamma \in \mathbb{Z}_p^{\ell \times (d'+1)}$ consists of the Lagrange coefficients for the shareholder evaluation points vs. the commitment, and $\vec{u} \in \mathbb{Z}_p^{\ell}$ are the shares received by that shareholder.

  Instead of checking that equality in full, the verifier can choose a random vector $\vec{r} \in \mathbb{Z}_p^{\ell}$ and verify that $\langle \vec{r}, \vec{u} \rangle \cdot G = \langle \vec{r} \cdot \Gamma, \vec{V} \rangle$. The dealer is disqualified (via a verifiable complaint) if that equality does not hold.

  If the dealer too holds multiple evaluation points (say m points), then it will send multiple vectors $\vec{V}$ and multiple vectors $\vec{u}$, which we can think of as matrices $V \in \mathbb{G}^{(d+1) \times m}$ and $\Psi \in \mathbb{Z}_p^{\ell \times m}$, respectively. The shareholder needs to verify the equality $\Psi \cdot G = \Gamma \cdot V$.

  To do it efficiently, they can choose two random scalar vectors $\vec{r} \in \mathbb{Z}_p^{\ell}, \vec{r'} \in \mathbb{Z}_p^{m}$ and verify $\vec{r} \Psi \vec{r'} \cdot G = \langle \vec{r} \cdot \Gamma, V \cdot \vec{r'} \rangle$. This way, a shareholder controlling ℓ evaluation point can verify the messages of a dealer with m points using only $(d'+1)(m+1)+1$ scalar-element products.

  Moreover, the vectors $\vec{r}, \vec{r'}$ above need not be uniform in $\mathbb{Z}_p$, to get statistical security of k bits it is enough to choose them as random k-bit numbers. When m is large, the vast majority of the scalar-element products use small scalars, reducing the complexity even further.
- Perhaps most importantly, the public group elements $R^{u,v}$ (that go into the hash for computing δ and the $e^{u,v}$'s) only need to be computed once for each shareholder, no matter how many evaluation points a party holds. As we explain in Section 2.10, this computation is asymptotically the most expensive part of the protocol, so it is important that no party needs to run it more than once.

A decryption service. In the blockchain setting, it is not hard to configure the parties running our protocols to also provide a decryption service, not just signatures. For safety, a decryption service should use a different secret key for decryption than for signatures. This can be done with a polynomial of degree one larger, using the techniques from Appendix I.

Once the committee has a Shamir sharing of the secret decryption key, they can directly use it to decrypt ElGamal-type ciphertexts: These ciphertexts include an element $R \in \mathbb{G}$, and to decrypt it is sufficient to recover $s \cdot R$ where s if the decryption key. The parties holding Shamir shares σ_i of s can just broadcast partial decryption shares $\sigma_i \cdot R$ (possibly with some proof of correctness), enabling anyone to compute $s \cdot R$ by "interpolating in the exponent."

B Faster Multiplication by a Super-Invertible Matrix

Recall that to compute the group elements $R_{u,v} = r_{u,v} \cdot G$ (which is needed in order to compute the Schnorr challenges $e_{u,v} = \mathsf{Hash}(S, R_{u,v}, M_{u,v})$), the signers must perform a matrix-multiplication "in the exponent",

$$\left[R_{u,v}\right]_{u\in[b],v\in[a]} \;=\; \Psi \times \left[\mathbf{H}_i(1-v)\cdot G\right]_{i\in\mathsf{QUAL},v\in[a]}. \tag{1}$$

We recall that Ψ needs to be super-invertible. Namely, any $b \times b$ sub-matrix of Ψ must be invertible.

We write $n = b + t$ in this section. We also assume $a = 1$ and ignore v and a indices everywhere: for $a > 1$, we just need to repeat the algorithm a times in parallel.

As explained in Section 2.4, we propose two solutions.

B.1 First Solution: Small-Scalar Vandermonde

Any Vandermonde matrix

$$\Psi = (\psi_j^{i-1})_{i\in[b],j\in[n]}$$

is super-invertible when the ψ_j's are distinct scalars, as shown in [26]. This is because a $b \times b$ submatrix of Ψ is itself a Vandermonde matrix which is invertible.

For faster multiplication, we can choose:

$$\psi_1, \ldots, \psi_n = -\lceil n/2 \rceil + 1, \ldots, \lfloor n/2 \rfloor$$

and then we can use a variant of the Horner's rule. Concretely, to evaluate Eq. (1), for each $j \in [n]$, we set $\phi_{1,j} = \mathbf{H}_j(1-v)\cdot G$, and compute $\phi_{i+1,j} = \psi_j \cdot \phi_{i,j}$ recursively for $i \in [b-1]$. Then, we have

$$R_u = \sum_{j=1}^{n} \phi_{u,j}.$$

Computing the opposite of a group element (from an elliptic curve), just consists in negating the y-coordinate which is extremely cheap. Therefore, the above algorithm essentially costs $(b-1)n$ scalar-by-element products with $\log n$-bit scalars. This is about $(b-1)n \log n / \log p$ full scalar-by-element products. That is a $\log n / \log p$ speed-up compared to the naive solution.

B.2 Second Solution: ECFFT-EXTEND

We now present our second solution which is asymptotically faster than the first, and also faster in practice for larger n.

B.2.1 From Super-Invertible to Smaller Hyper-Invertible

We first remark that it is beneficial to make the matrix $\Psi \in \mathbb{Z}_p^{b\times(b+t)}$ as sparse as we can, while ensuring that it remains super-invertible. To that end, we use the construction

$$\Psi = (I|H),$$

where I is the $b \times b$ identity matrix and H is a $b \times t$ *hyper-invertible* matrix [3]. Recall that H is hyper-invertible if every square sub-matrix of it is invertible (not just any $b \times b$ sub-matrix).

Lemma 1.1. If H is a hyper-invertible $b \times t$ matrix and I_b is the $b \times b$ identity matrix, then $\Psi = (I_b|H)$ is super-invertible.

Proof. Consider any $b \times b$ sub-matrix $\Psi' = (I'|H')$ of Ψ, it consists of some number k of columns from I and $b-k$ columns from H. By swapping rows we can move all the single 1's from I' to the top k rows, and then by column reduction, we can zero out all the other entries in these k top rows without affecting any of the $b-k$ bottom rows. This does not change the rank, but results in a matrix of the form

$$\Psi'' = \left[\begin{array}{c|c} I_k & 0 \\ \hline 0 & H'' \end{array}\right],$$

where I_k is the $k \times k$ identity matrix and H'' is some $(b-k)\times(b-k)$ sub-matrix of H. Since H is hyper-invertible then H'' is invertible, and therefore so is Ψ'' and therefore Ψ'. □

We remark that this matrix Ψ is "as sparse as possible", in that no row of a super-invertible matrix can have more than $b-1$ zeros.

Using this construction, we can already reduce the number of scalar-by-element products when computing Eq. (1) to only $b \cdot t$ rather than $b \cdot (t+b)$, even when using the naive matrix-multiplication algorithm. (Recall we assume $a = 1$ here.) However, for most parameter regimes, the Vandermonde solution from Appendix B.1, would be faster than just using $\Psi = (I|H)$ with an arbitrary hyper-invertible matrix.[18]

B.2.2 Hyper-Invertible Matrix from ECFFT EXTEND

The previous subsection showed that we just need to find a hyper-invertible matrix H, for which multiplication by a vector (on the right) is faster than naive matrix-vector multiplication.[19] We remark that any submatrix of a hyper-invertible matrix is hyper-invertible. So we can focus on the case of square hyper-invertible matrices $H \in \mathbb{Z}_p^{k\times k}$. (We will take $k = \max(b,t)$.)

Let us now recall the hyper-invertible matrix construction from [3]. Let $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k \in \mathbb{Z}_p$ be $2k$ distinct scalars. We define the hyper-invertible matrix H as follows: $\vec{y} = H \cdot \vec{x}$ is defined as follows. Let f be the unique degree-$(k-1)$ polynomial such that $f(\alpha_i) = x_i$ for $i \in [k]$. Then $y_i = f(\beta_i)$ for $i \in [k]$.

[18] Exception would be for very low number of dishonest parties, e.g., $t = 1$ and $b \gg 1$.

[19] Note that Vandermonde matrices are not hyper-invertible in general. For example, the 3×3 Vandermonde (for the scalars $-1, 0, 1$) is

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & 0 & 1 \end{pmatrix}$$

and it contains the 2×2 square matrix with only 1, which is not invertible.

Now, we remark that the ECFFT EXTEND algorithm from [5] is actually doing the exact above computation for $(\alpha_1, \ldots, \alpha_k)$ and $(\beta_1, \ldots, \beta_k)$ two "i-basic sets" (as defined in [5], for the appropriate i), when k is a power of 2.

We also remark that the ECFFT EXTEND algorithm is linear in its inputs ($\vec{x}$ with the notation above), which means we can use it to evaluate the matrix multiplication "in the exponent" as required for Eq. (1). The fact that the inputs are group elements is actually helpful in our case: scalar-by-element products are much slower than scalar-by-scalar products and hence the overhead of the ECFFT algorithm does not impact our application as much as it impacts uses over a field. We show that ECFFT EXTEND is actually competitive in Appendix B.3

ECFFT EXTEND only works for k, a power of two. If k is not a power of 2, we can just consider the next power of 2, padding the input with zeroes and removing rows of the output.

Note that if $\mathbb{Z}_p$ is an FFT-friendly field with a large-power-of-2 root of unity, then it would be faster to use FFT-base algorithms than ECFFT EXTEND. FFT can indeed be adapted to compute the above EXTEND algorithm. However, in many practical settings, we do not have the choice of the elliptic curve, and the associated scalar field $\mathbb{Z}_p$ is not FFT-friendly. In particular, the scalar field of ED25519 is not FFT-friendly.

B.3 Benchmarking

For the benchmark, we consider the case where $b = t = k = 2^\ell$ for some ℓ. In particular, the matrix H for the second solution is square and of size, a power of two. If this was not the case, we would need to execute the ECFFT EXTEND algorithm for k being the next power of 2 that is larger or equal to $\max(b, t)$. The input would be padded with zeroes and the extra output values would be ignored.

We focused on the ED25519 curve. The order of $\mathbb{Z}_p$ is $p = 2^{252} + 27742317777372353535851937790883648493$. To apply the ECFFT EXTEND algorithm for $k = 2^\ell$, we need to find an elliptic curve over $\mathbb{Z}_p$ with order divisible by $2^{\ell+1}$.[20] We found an elliptic curve suitable for $k \leq 2^{14}$ in a few hours with a naive Sage code on a laptop. The elliptic curve found is:

$$y^2 = x^3 + ax + b$$

with $a = 43929768024911012771198582337486288866704470977431655735539794616091255506 24$ and $b = 26413901165040580465939823648559350546241969132029613551899675222922222358134$.

We implemented both the ECFFT EXTEND solution and the small-scalar Vandermonde solution. The code is written in Rust and based on the Rust crate/library ecfft-bn254 [7] and Arkworks [2]. It is not fully optimized but already shows that the algorithms are practical. In particular, the use of more optimized ED25519 libraries and faster double-scalar-multiplication algorithms are expected to provide additional speed-up. The code is single-threaded. In the full protocol, the matrix multiplication is executed a independent times, which is trivial to parallelize, improving even more overall efficiency.

The benchmarking code is available from `https://github.com/fabrice102/ecfft-group`, under the MIT license. This code is based on the code [7] and adapts it to work with polynomials with coefficients in a group, instead of in the base field.

[20] Note that this elliptic curve is completely unrelated to ED25519. It is not even on the same base field. Furthermore, because of the way the ecfft-bn254 library [7] was written, we actually need the created elliptic curve to have a *cyclic* subgroup of order 2^ℓ, instead of just having a subgroup of order divisible by 2^ℓ.

Benchmarks results are provided in Table 1 and were run on a VM with 2 CPU Cores (AMD EPYC 7601, 2.2GHz) and 4GB RAM (OS: Ubuntu 20.04.3 LTS, rust version 1.69.0, 2023-04-16). The Vandermonde solution timings are extrapolated from the benchmark code that computes Horner's evaluation on small scalars: the time of evaluating the matrix multiplication with Horner's rule variant is the same as evaluating a polynomial of degree $2b-1$ in $2b$ small points (with Horner's rule), multiplied by $b - 1/(2b - 1)$ (since our degree is $b - 1$). We compare to the naive solution (i.e., naive matrix multiplication by an arbitrary super-invertible matrix without any optimization) and to the "multi-scalar multiplication" naive solution where each element of the resulting vector is computed using the multi-scalar multiplication algorithm implemented in Arkworks [2] (Pippenger algorithm), which yields a $\log n$ improvement asymptotically. The naive solution timings are extrapolated from the timing of a single scalar multiplication, multiplied by $b \cdot n = 2k^2$, while the multi-scalar multiplication solution timings are extrapolated from a single multi-scalar multiplication in dimension $n = 2k$, multiplied by $k = b$. We remark that both our solutions outperform the naive and the multi-scalar multiplication solutions with Pippenger for all the parameters b considered.

Table 1: Benchmark of Super-Invertible Matrix Multiplication
(see text in Appendix B.3 for notes on how the timings were computed)

$b = t$	Naive	Multi-scalar multiplication	Small Vandermonde	ECFFT EXTEND
2^1	1.26 ms	1.83 ms	4.22 us	978 us
2^2	5.04 ms	4.34 ms	35.7 us	4.66 ms
2^3	20.1 ms	11.2 ms	222 us	14.0 ms
2^4	80.6 ms	39.0 ms	1.23 ms	39.0 ms
2^5	322 ms	132 ms	6.08 ms	98.9 ms
2^6	1.29 s	348 ms	30.0 ms	238 ms
2^7	5.16 s	1.20 s	140 ms	563 ms
2^8	20.6 s	4.19 s	630 ms	1.27 s
2^9	82.5 s	12.8 s	2.87 s	2.85 s
2^{10}	330 s	46.2 s	12.7 s	6.37 s
2^{11}	1,320 s	168 s	56.7 s	14.0 s
2^{12}	5,282 s	572 s	249 s	30.7 s
2^{13}	21,126 s	2,021 s	1,083 s	68.3 s
2^{14}	84,506 s	7,617 s	4,702 s	146 s

C Parameters and Performance

C.1 The parameters n, t, a and b

For given parameters t and a, the ephemeral secrets $r^{u,v}$ are shared via polynomials of degree $t + 2a - 2$, so we need at least $t + 2a - 1$ honest shareholders to obtain shares in order to be able to use them. All these shareholders must be in HOLD, and there could be up to t corrupted shareholders there, so the agreement protocol must ensure that HOLD is of size at least $2t+2a-1$. As there could be t corrupted parties during the run of the agreement protocol, then to ensure $|\mathsf{HOLD}| \geq 2t+2a-1$ the number of shareholders must be $n \geq 3t + 2a - 1$ (for both the static-committee setting and the

dynamic setting). Below we mostly assume $n = 3t + 2a - 1$.

For the parameter b, recall that we need b honest dealers in QUAL for the DKG protocol. We can ensure QUAL as large as $n - t$ (but of course not any larger), so the largest value that we can ensure for b is $b = n - 2t$. The number of signatures in each run of the protocol is therefore $a \cdot b = a(n-2t)$. We note, however, that at the end of each run of the agreement protocol, everyone knows the size of QUAL, and can set the parameter b accordingly. If QUAL happens to be larger than $n - t$ in a particular run, then b can be larger than $n - 2t$, so we can sign more messages in that run. In the best case, we have $|\mathsf{QUAL}| = n$, so we can sign $a(n - t)$ messages.

C.2 Bandwidth in the static-committee setting

In the static setting we do not need to re-share the long-term secret key, so each dealer only shares a single random polynomial $\mathbf{H}_i$ of degree $d' = t + 2a - 2$. This means broadcasting $t + 2a - 1$ group elements for the commitments, and n ciphertexts encrypting the shares. Assuming ElGamal encryption (with randomness re-use), encryption takes $n + 1$ more group elements. The total per dealer is therefore $n + t + 2a$, for a grand total of $n(n + t + 2a)$ group elements.

In the optimistic case where (almost) everyone is synchronized, the agreement protocol requires a single broadcast from each shareholder (after the dealers sent their messages). If all the dealers are good then these messages are short, otherwise, each message can include up to t complaints, each with just a few group elements, for an additional bandwidth of $O(nt)$.

For the signature generation part, we have $|\mathsf{HOLD}| = n - t = 2a + 2a - 1$ and each shareholder in HOLD broadcasts $b = n - 2t$ shares, for a total broadcast bandwidth of $(2t + 2a - 1)(n - 2t)$ scalars, enabling the generation of up to $a(n - 2t)$ signatures.

The total bandwidth (both group elements and scalars) is therefore $n(n + t + 2a) + (2t + 2a - 1)(n - 2t) = O(n^2)$, plus at most $O(t^2)$ for all the complaints. The amortized bandwidth per signature (not counting complaints) is about

$$\frac{n \cdot (n + t + 2a) + (2t + 2a - 1) \cdot (n - 2t)}{a \cdot (n - 2t)} \quad = \quad \frac{n^2 + nt}{a(n - 2t)} + \frac{2n}{n - 2t} + \frac{2t - 1}{a} + 2.$$

Asymptotically, setting $t = a = \Omega(n)$ we get $O(1)$ group-elements/scalars per signature.

For a few examples, setting $t = a = n/5$ we get about 17.33 scalars/group-elements per signature. To get resilience of $t = n/4$ we can only get $a = n/8$, yielding about 34 scalars/group-elements per signature. In the other direction, reducing the resilience to $t = n/10$ we can set $a = 7n/20$, getting an amortized 9 scalars/group-elements per signature. (That is 4.5 times the bandwidth needed to communicate just the signatures themselves.)

C.3 Computation in the static-committee setting

In terms of computation (and counting only scalar-by-element products), each dealer needs to compute $d' = t + 2a - 2$ products to commit to their polynomial and $n + 1$ more for the encryptions.

Each shareholder must verify all the shares that they received, each costing about $d' + 1$ products, so $n \cdot (d' + 1)$ products for each shareholder. For complaints, each shareholder needs to generate at most t of them, and verify at most $2t$ of them, since at most t dealers and t shareholders can be bad. (In the static committee case, only t complaints need to be verified.) Verifying each complaint can take another $d' + O(1)$ products (d' to compute the commitment and $O(1)$ for the proof of

decryption.) Hence, the agreement can take up to (about) $(n+t)(d'+1)$ products for each party, regardless of how many messages will be signed.

For generating the signatures, the shareholders must first compute all the $R^{u,v}$'s to be hashed, then compute their own shares. By our convention, commitments to the $\mathbf{H}_i$'s include in particular the elements $\mathbf{H}_i(1-v)\cdot G$ for $v \in [a]$. Hence, for any $v \in [a]$, computing the elements $R^{u,v} = \mathbf{H}^u(1-v)$ for all $u \in [b]$ requires multiplying the vector $(\mathbf{H}_i(1-v) : i \in \mathsf{QUAL})$ by the super invertible matrix Ψ of dimension $b \times (b+t)$. Doing it for all $v \in [a]$ means computing the matrix product

$$\left[R^{u,v}\right]_{u\in[b],v\in[a]} \;=\; \Psi \times \left[\mathbf{H}_i(1-v)\cdot G\right]_{i\in\mathsf{QUAL},v\in[a]}.$$

Using the Vandermonde solution from Section 2.4, this would take about $a\cdot b\cdot t \log n/\log p$ products. Using the ECFFT EXTEND solution, this would take $O(a\cdot \max(b,t)\log(\max(b,t)))$, which in practice is as fast as the first solution when $b=t=512$, and becomes faster afterwards.

Once, everyone broadcasts their shares, verifying the shares and assembling the signature can be done by anyone who sees the broadcast channel, not necessarily the parties themselves, and certainly not all parties need to carry out that verification. [21] Verifying each share takes $d'+1$ products, and at most $d'+t+1$ of them need to be verified before we have $d'+1$ valid ones, for a total of $(d'+1)(d'+t+1)$ products, but since not every party needs to carry it out we do not include these products in the tally below.

Substituting $d' = t+2a-2$ and $b = n-2t$, the overall number of scalar-by-element products for each party is therefore $(n+t+2a-1)+(n+t)(t+2a-1)+a(n-2t)t\log n/\log p$ products (with the small Vandermonde solution from Section 2.4), and the per-signature number is

$$\begin{aligned}&\frac{(n+t+2a-1)+(n+t)(t+2a-1)+a(n-2t)t\log n/\log p}{a(n-2t)}\\ &= t\log n/\log p + \frac{(n+t)(t+2a)}{a(n-2t)} + \frac{2-\frac{1}{a}}{n-2t}.\end{aligned}$$

With $a=\Omega(n)$ and $n-2t=\Omega(n)$, and when the Vandermonde solution from Section 2.4, this yields complexity of $t\log n/\log p + O(1)$ products per signature. (In the example of $t=a=n/5$, we get about $t+6$ products for each signature.) With $a=\Omega(n)$ and $n-2t=\Omega(n)$, and when the ECFFT EXTEND solution from Section 2.4, this yields complexity of $O(\log n)$ products per signature.

C.4 Performance in the dynamic/proactive setting

In this setting, we also need to refresh the long-term secret. This means that each dealer broadcasts $n+t$ additional group elements, and shareholders may need to broadcast more complaints, but the bandwidth for signature generation remains unchanged. Therefore, the overall bandwidth in this case (not counting complaints) becomes $2n(n+t+a)+(2t+2a-1)(n-2t)$, increasing the per-signature bandwidth by $\frac{n(n+t)}{a(n-2t)}$. (For example, when $t=a=n/5$, the bandwidth increases from 17.33 to 27.33 scalars/group-elements per signature.)

Computation likewise increases much less than 2×. Each dealer performs only t additional products for the commitments.[22] Verifying the sub-shares takes only $n(t+1)$ more products by each shareholder, and checking each complaint is only $t+O(1)$ more products.

[21] Verification can even be avoided in the optimistic case, by just trying to reconstruct a polynomial and see that (almost) all the shares agree with it.

[22] Using hybrid encryption does not take any more products to encrypt longer messages.

C.5 Some numerical examples

The techniques in this paper are useful even for fairly small committees. For example:

- With $n = 10$, we need $t \leq 3$ even to sign just a single message, but setting for $t = 2$ allows us to set $a = 2, b = 6$ and sign $a \cdot b = 12$ messages for the price of a single message. That's a $12\times$ performance improvement for a small drop in resilience.
- For $n = 16$ and $t = 3$ we can set $a = 4, b = 10$ and generate 40 signatures.
- For $n = 64$ and $t = 15$ we can set $a = 10, b = 34$ and generate 340 signatures.
- In Appendix D.1 we analyze the committee sizes that we need when sub-sampling the committees in a few settings. For example, when assuming 80% honest majority in the overall population and shooting for a 2^{-80} probability of safety failure and 2^{-11} probability of liveness error due to sub-sampling, we can use committees of size $n = 992$ with $t = 336$ and $a = 40$ (and $b = 320$). We note that here we have $n < 3t + 1$, since we use different thresholds for liveness and safety errors.

  In this setting, we can sign $40 \cdot 320 = 12800$ messages per run, and each run consumes broadcast bandwidth of $2n(n + t + a) + (2t + 2a - 1)(n - 2t) = 2,954,432$ scalars and group elements, or about 95 megabytes. In terms of computation, each party (playing first a shareholder and then the dealer in the next epoch) needs to perform about $(n + 2t + 2a - 1) + (n + t)(t + 2a - 1) + n(t + 1) + a(n - 2t)t \log n / \log p = 1,055,167$ scalar-element products (for $\log p = 256$. Moreover, most of the products are actually dot product between a vector of scalars and a vector of group elements, which can be done perhaps $3\times$ faster than performing each product separately. On contemporary servers, even a single-threaded implementation can perform this number of products in under three minutes, yielding an amortized rate better than 4000 signatures per minute.
- Running the protocol with optimistic parameters, we may try for a setting that still provides 2^{-80} safety error with 80% honest but liveness-failure probability of 2^{-8} with 95% honest. To get roughly the same number of signatures per run we set $a = 64$, thus getting $n = 676$, $t = 250$, and $b = 176$. This yields more or less a $2\times$ lower complexity, with a bandwidth of 1,448,832 scalars/group-elements and 630,081 scalar-by-element products, while producing 11264 signatures. Moreover, in the even-more-optimistic case where all the dealers happen to be honest, we can sign as many as $a(n - t) = 27264$ messages with the same parameters, another $2\times$ improvement (and a rate of more than 10,000 signatures per minute).

D Deploying in a Blockchain Environment

Next we describe how the protocols from above can be used to implement a large-scale Schnorr-signature service over a public blockchain. Such implementation would use all the techniques from Section 2 and Appendix A. Specifically,

- It would use the QUAL-agreement protocol from Appendix E, where the blockchain serves as the underlying total-order broadcast channel;
- It includes the proactive share refresh from Section 3.2, which is run periodically every few blockchain rounds (called an epoch);

- The committees in each epoch are sub-sampled from among the validators using the randomness beacon as discussed in Section 2.8;
- It would include all the optimizations from Appendix A, such as optimistic runs and user aggregation.

Below we discuss a few other aspects of such implementation, such as the required committee sizes, how to use the single blockchain key s to sign on behalf of multiple smart contracts, and plausible mechanisms for recovering from catastrophic failures.

D.1 Committee Sizes

We assume a huge network, consisting of many millions of parties, from which we sub-sample the committees. We note that this model is sometimes applicable even for a blockchain with only a handful of validators. Specifically, in a proof-of-stake blockchain we may want to give high-stake nodes more power in the computation than low-stake ones, to align the trust model of the signature service with that of the underlying consensus protocol. (E.g., both the consensus protocol and the higher-level signature scheme remain secure as long as at least 80% of the stake is controlled by honest parties.)

In that case, we may want to view each token as a party, where a physical node is running the protocol on behalf of all the tokens that it controls. Even if the network only has a few dozen physical nodes, this view may require that we use our protocols with many millions of virtual parties.[23]

Below we denote the overall number of parties by N, and we assume a PKI where we have a list of all N parties, each with their public encryption key. Also set f be (an upper bound on) the fraction of corrupted parties, namely we assume that we have at most fN corrupted parties and at least $(1-f)N$ honest ones.

Recall that we sub-sample each committee based on a (pseudo)random value from the randomness beacon, expanded using a PRG. In more detail, for some parameter n (to be determined below), we expand the latest randomness beacon value U into a (pseudo)random vector of n indexes in $[N]$ (with repetitions), $PRG(U) = (i_1, i_2, \ldots, i_n) \in [N]^n$. The next committee then consists of the n parties that are indexed by $i_1, \ldots, i_n$ in the PKI.[24]

Chosen this way, the committee size will be exactly n, and the number of corrupted parties in the committee will be upper-bounded by $X_C \sim \mathsf{Bin}_{n,f}$, a Binomial random variable with parameters n, f. Similarly, the number of honest parties in the committee is lower-bounded by $X_H \sim \mathsf{Bin}_{n,1-f}$. Conveniently, these two random variables do not depend on N, the total number of parties in the system.

The parameters n, t will be chosen based on a and f (and the required safety and liveness guarantees $\varepsilon_{\text{safety}}, \varepsilon_{\text{liveness}}$), to ensure the following conditions:

Safety. To prevent the corrupted parties from learning the secret key, we need $\Pr[X_C > t] \leq \varepsilon_{\text{safety}}$.

Liveness. We need $\Pr[X_H < 2t + 2a - 1] \leq \varepsilon_{\text{liveness}}$ to ensure that honest parties can reconstruct the signatures.

[23] This is also the setting where the user-aggregation ideas from Appendix A will have the most impact.

[24] If an index $i \in [N]$ appears more than once in the vector, then the corresponding party will have more than one seat on the committee and will get more than one share of the relevant secrets.

To find suitable parameters, we therefore wrote a simple program that takes as input $f, a, \varepsilon_{\text{safety}}$, $\varepsilon_{\text{liveness}}$, and searches for the smallest values for n, t that satisfy these two conditions. Once those parameters are set, we instantiate the system with a QUAL-agreement protocol from Fig. 3 with parameters $d_0 = d_1 = n - t$ and $d_2 = t + a$. This implies a parameter b (the dimension of the super-invertible matrix) which is $b = |\mathsf{QUAL}_1| - t = n - 2t$.

D.1.1 Optimistic Parameters

As mentioned in Appendix A, we can also attempt to run with optimistic parameters, i.e., smaller committee, as long as it is only liveness that we sacrifice, not safety. For that purpose, we modified the parameter-searching program to take another set of parameters $f', \varepsilon'_{\text{liveness}}$, then define $X'_H \sim \mathsf{Bin}_{n,1-f'}$ and search also for parameters that satisfy $\Pr[X_C > t] \leq \varepsilon_{\text{safety}}$ and $\Pr[X'_H < 2t+2a-1] \leq \varepsilon'_{\text{liveness}}$.

D.2 Uses of Additive Key Derivation

Additive key derivation (cf. [21]) allow a single secret key to be used to sign on behalf of multiple public keys. Let s be a "master secret key" with corresponding public key $S = s \cdot G$. Then a derived key-pair can be specified by a public "tweak" $u \in \mathbb{Z}_p$, which defines the secret key $s' = s + u$ and the public key $S' = S + u \cdot G$. This is a well known technique, which is used in many blockchains (including Bitcoin [41]). To generate a Schnorr signature on message M relative to the derived key S', one can use the master secret key in conjunction with the public tweak.

In the context of a blockchain signature service, we consider a blockchain master key whose secret key is shared among the validators as described in this work. We can then give each smart contract its own derived key, by setting the tweak value u to be (say) a hash of the smart contract identity I (or code): $u = \mathsf{Hash}(I)$. This way, each smart contract "owns" its own key, and can ask the blockchain to sign messages relative to that key. When a smart contract with identity I asks to sign a message M, the validators will execute the Schnorr signature protocol exactly as specified in this work, except that they use the public scalar $e = \mathsf{Hash}(S', R, M)$ for that message (instead of $e = \mathsf{Hash}(S, R, M)$). This will result in a pair $(R, es + r)$, that can be converted to standard Ed25519 signature by adding $e \cdot \mathsf{Hash}(I)$ to the second entry in the pair.

D.3 Recovery from Catastrophic Failures

In the proactive setting that we consider, the system relies on adequate connectivity to make progress and refresh the secret key from one committee to the next. A plausible attack vector is mounting a network partition attack. This will stall progress and deprive the parties of the ability to refresh the sharing and erase their old shares. If the outage lasts for a long time, it may become necessary for nodes to delete their old shares without refreshing them, for fear that a long-held secret may become an easy target for an attack. If that happens, how can the system recover and return to normal operation once the outage is over?

One recovery approach is to fall back on centralized trust: the master key can be kept in (very) cold storage, perhaps shared among a handful of highly trusted parties, and recovered from them in the event of such a devastating attack. This solution, however, has the drawback that these "highly trusted" parties may not be trustworthy after all: They can recover the key even with no attack, and there would not even be any way of knowing if they did it.

A somewhat better approach would be to equip those highly trusted parties with an independent recovery decryption key, with the corresponding public key embedded in the blockchain code. This key is never used, except when a shareholder needs to delete their share without being able to pass it forward. In that situation, the shareholder will encrypt their share under the recovery public key before deleting it from memory. Once the outage is over, they will run a special recovery protocol, in which the highly trusted parties help them to recover the share and continue where they left of. It is even possible to implement a hybrid approach, where some small number of shares are always encrypted under the public key, but most other shares are only encrypted when an attack happens. This way the highly trusted parties can fill in for some shareholders that lost interest after the attack and did not participate in the recovery process.

Yet another approach, which relies on anonymous public-key encryption, is as follows: When a shareholder needs to delete their share without being able to pass it forward, it chooses a random committee and secret-share its share to them, broadcasting an encryption of the sub-shares under their anonymous-PKE keys. This way, the adversary does not know who is holding what shares, but the parties can still recover those shares when the system resumes.

E The Full Agreement Protocol

Here we describe the full agreement protocol, the one that we need to use for the dynamic/proactive setting. Recall that the main difference between this and the base protocol from Section 4 is that the latter only ensures $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$, whereas here we need to ensure $|\mathsf{QUAL}| \geq d_1$. Also, here we need to output a single shareholder subset HOLD but two pairs of dealer subsets $(\mathsf{QUAL}_1, \mathsf{BAD}_1)$ and $(\mathsf{QUAL}_2, \mathsf{BAD}_2)$. The full protocol consists of multiple iterations, each time running the base protocol from Fig. 3. Each iterations adds to the dealer-subsets from the previous iteration, and re-defines the set HOLD "from scratch".

This protocol has two sets of dealers (not necessarily disjoint), one sends messages related to $\mathbf{H}$ and the other messages related to $\mathbf{F}$. Below we refer to them as "type-1" and "type-2" messages. The protocol is described in Fig. 5, and its correctness is stated in Theorem 2 and proved in Appendix F.

Theorem 2. Consider an execution of the base agreement protocol from Fig. 5 over a total-order broadcast channel, among a set of n_0 shareholders of which at least d_0 are honest. Assume that at most n_1, n_2 dealers broadcast type-1,type-2 messages, respectively, at least d_1, d_2 of these dealers are honest, and no verifiable complaint can be constructed against any honest dealer. Then all honest shareholders will eventually terminate, all outputting the same sets with $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}_k| \geq d_k$ for $k = 1, 2$. Moreover:

- No shareholder in HOLD complained against any dealer in any QUAL_k; and
- Every dealer in BAD_k has at least one shareholder (not necessarily in HOLD) that lodged a verifiable complaint against them.

E.1 Optimizations and Variants

One obvious optimizations of this protocol is that parties can remember messages that were seen on the broadcast channel in previous iterations rather that re-sending them again.

We note that the main drawback of this protocol as compared to the one from Fig. 3 is that shareholders need to broadcast multiple complaint rounds (at most t of them if there are t dishonest

Parameters: $n_0, d_0, n_1, d_1, n_2, d_2$.

Precondition: We have two sets of dealers for type-1 (resp. type-2) messages, of sizes at most n_1 (resp. n_2), at least d_1 (resp. d_2) of which are honest. We also have n_0 shareholders, at least d_0 of them are honest.

Shareholder P_j:

Initialize $\mathsf{HOLD} = \mathsf{QUAL}_1 = \mathsf{BAD}_1 = \mathsf{QUAL}_2 = \mathsf{BAD}_2 = \emptyset$. Then repeat:

1. **Enlarge QUAL's until $|\mathsf{QUAL}_k| \geq d_k$ for $k = 1, 2$.** When receiving the (first) broadcast message of type-k from dealer D_i, set $\mathsf{QUAL}_k := \mathsf{QUAL}_k \cup \{D_i\}$. Keep adding until both QUAL_k sets reach the required size.

2. **Broadcast Complaint.** Wait until $|\mathsf{QUAL}_k| \geq d_k$ for $k = 1, 2$. Then broadcast all the verifiable complaints against dealers in $\mathsf{QUAL}_1, \mathsf{QUAL}_2$ whose messages were locally bad, in a single broadcast message. If this set is empty, broadcast the empty set.

3. **Contract QUAL's.** Collect all the valid complaint-sets (i.e., the ones whose complaints can be verified). Once there are d_0 valid complaint-sets, set $\mathsf{QUAL}_k := \mathsf{QUAL}_k \setminus \{D_i\}$ and $\mathsf{BAD}_k := \mathsf{BAD}_k \cup \{D_i\}$ for each verifiable complaint against a type-k message from dealer D_i;

4. **Set HOLD.** To the first d_0 shareholders who broadcasted a valid complaint-set in this iteration.

If $|\mathsf{QUAL}_k| \geq d_k$ for $k = 1, 2$, output $\mathsf{HOLD}, \mathsf{QUAL}_1, \mathsf{BAD}_1, \mathsf{QUAL}_2, \mathsf{BAD}_2$ and halt.
Otherwise go back to step 1 (and use any new dealer messages that were received since last time).

Figure 5: The full agreement protocol

dealers). We also note that we can have protocols in between the ones from Fig. 3 and Fig. 5: For a parameter $\tau \in [1, t]$, we can limit the number of complaint rounds to t/τ by changing the halting condition, checking that If $|\mathsf{QUAL}_k| \geq d_k - \tau$ (rather than $|\mathsf{QUAL}_k| \geq d_k$). This way, the sets BAD must grow by at least τ in each iteration (except the last one), so the number of iterations cannot be more than $1 + t/\tau$.

F Proofs of Theorems 1 and 2

Theorem 1. *Consider an execution of the base agreement protocol from Fig. 3 over a total-order broadcast channel, among a set of n_0 shareholders of which at least d_0 are honest. Assume that at most n_1 dealers broadcast messages, at least d_1 of these dealers are honest, and no verifiable complaint can be constructed against any honest dealer. Then all honest shareholders will eventually terminate, all outputting the same sets with $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$. Moreover:*

- *No shareholder in HOLD complained against any dealer in QUAL; and*
- *Every dealer in BAD has at least one shareholder in HOLD that lodged a verifiable complaint against them.*

Proof. Recall that we refer to shareholders that read τ messages from the channel as being "in step τ".

Agreement is easy to verify: The sets $\mathsf{QUAL}, \mathsf{BAD}, \mathsf{HOLD}$ that a shareholder maintains throughout the protocol are just deterministic functions of the messages on the broadcast channel. Hence, at any given step, all shareholders at a given step will have the same sets. Since the halting condition is also a deterministic function of the broadcast channel then they will all halt at the same step and therefore all output the same sets $\mathsf{QUAL}, \mathsf{BAD}, \mathsf{HOLD}$.

It can be verified by inspection that when that happens, we have $|\mathsf{HOLD}| = d_0$ and $|\mathsf{QUAL}_k| + |\mathsf{BAD}| \geq d_1$. Also it is clear by inspection that shareholders in HOLD never complain about dealers in QUAL, and that every dealer in BAD was moved there as a result of some verifiable complaint.

The only thing left to show is termination. Since there are at least d_1 honest dealers, then by eventual delivery we know that all the honest shareholders will eventually receive d_1 dealer messages and thus they will all complete Step 1 of the protocol (enlarge QUAL), and then broadcast a message in Step 2 (broadcast complains).

Similarly, since there are at least d_0 honest shareholders and all of them will eventually broadcast a message in Step 2, then all honest shareholders will eventually receive at least d_0 valid complaint-step, and therefore fix HOLD and halt. □

Theorem 2. *Consider an execution of the base agreement protocol from Fig. 5 over a total-order broadcast channel, among a set of n_0 shareholders of which at least d_0 are honest. Assume that at most n_1, n_2 dealers broadcast type-1,type-2 messages, respectively, at least d_1, d_2 of these dealers are honest, and no verifiable complaint can be constructed against any honest dealer. Then all honest shareholders will eventually terminate, all outputting the same sets with $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}_k| \geq d_k$ for $k = 1, 2$. Moreover:*

- *No shareholder in HOLD complained against any dealer in any QUAL_k; and*
- *Every dealer in BAD_k has at least one shareholder (not necessarily in HOLD) that lodged a verifiable complaint against them.*

Proof. Agreement is argued exactly as for Theorem 1, since all the sets and the halting condition are deterministic functions of the broadcast channel then all honest parties will agree on them.

It is also easy to verify by inspection that honest parties halt only when $|\mathsf{QUAL}_1| \geq d_1$ and $|\mathsf{QUAL}_2| \geq d_2$. Similarly it is clear that no shareholder in the final HOLD complain about any dealer in $\mathsf{QUAL}_1, \mathsf{QUAL}_2$ (or else these dealers will be removed from their QUAL), and that some shareholder must have lodged a verifiable complaint against every dealer in $\mathsf{BAD}_1, \mathsf{BAD}_2$ (since that's the only way that dealers are added to the BAD's.).

For termination, we prove by induction over the iterations that all honest parties will complete Step 1 in each iteration (until the protocol terminates). Consider iteration $i-1$, that ended with some sets $\mathsf{BAD}_1, \mathsf{BAD}_2$ of sizes b_1, b_2, respectively. Since there are at least d_1, d_2 honest dealers in the two dealer sets (and they cannot be in the BAD sets), then we know that all honest shareholders must eventually receive at least that many broadcast messages from dealers outside of these BAD sets. When that happens for the first time, those dealers will be included in shareholder will in $\mathsf{QUAL}_1, \mathsf{QUAL}_2$ as maintained by the honest shareholders, and therefore those shareholders will terminate Step 1 of the next iteration and broadcast their Step 2 message for this iteration. Then, all the honest shareholders will eventually get these messages and complete Step 3 and 4 of that iteration. This completes the proof that each iteration will eventually terminate.

Next we prove that the number of iterations is at most $(n_1 - d_1) + (n_2 - d_2) + 1$. To see that, note that for each iteration, the sizes of $\mathsf{QUAL}_1, \mathsf{QUAL}_2$ after Step 1 are at least d_1, d_2, respectively. So if no more dealers are added to the BAD sets then the protocol will terminate after that iteration. It follows that in all but the last iteration, at least one of $\mathsf{BAD}_1, \mathsf{BAD}_2$ must grow. Moreover, only dishonest dealers can ever be added to these BAD sets. As there are at most $(n_1 - d_1)$ dishonest dealers in the first set and at most $(n_2 - d_2)$ dishonest dealers in the second set, the total number of iterations cannot be larger than $(n_1 - d_1) + (n_2 - d_2) + 1$. □

G Security Proof of the Threshold Signature Protocol

We now turn to proving the security of our base threshold signature protocol for the static-committee setting from Fig. 1. We build the proof step by step, starting from the proof for the (centralized) Schnorr signature scheme [37] and a simple threshold Schnorr signature protocol a-la-GJKR, adapting their proof techniques and adding components as needed for our protocols. Specifically, we define a list of variants of the protocol, and explain how the proof is modified from one variant to the next:

1. Centralized Schnorr (Appendix G.1): Pointcheval and Stern [37] proved that, under the discrete log assumption, Schnorr signature scheme is secure in the random oracle model. The key component of their proof is the forking lemma [37, Theorem 10], which we will also use for our proof in the following variants.

2. Threshold Schnorr for a single message (Appendix G.2): This is similar to (but not exactly the same as) the GJKR protocol from [17, Fig.4], using Feldman commitments. While using Feldman allows a rushing adversary to bias the distribution of the ephemeral randomness, Gennaro et al. proved in [17] that their threshold signature protocol with Feldman commitments is still secure. We prove the same for our protocol by adapting their techniques to our needs, see Appendix G.2.

3. Parallel Threshold Schnorr signatures (Appendix G.3): It is known that the GJKR proof for the single-signature threshold scheme does not extend to signing multiple messages in parallel, in fact the resulting scheme is insecure. We therefore add a mitigation technique that allows us to recover the security argument when signing a set of messages in parallel as our protocol does.

4. Non-Packed threshold Schnorr with a super-invertible matrix (Appendix G.4): Our use of a super-invertible matrix allows each dealer to shares only a single polynomial, but we can still derive multiple signatures. The security proof for this variant requires a generalization of the simulation technique, as well as a small change to the protocol itself.

5. Threshold Schnorr with super-invertible matrix and packing (Appendix G.5): This part requires another generalization of the simulation technique.

Combining all these techniques, we describe the final reduction in Appendix G.6. Finally, in Appendix G.7 we then discussed the small changes for the dynamic/proactive setting. We begin with some preliminaries.

Assumption 1 (The Discrete-Logarithm Assumption)**.** Let $\mathbb{G}$ be a group of prime order $p \in \mathsf{PRIME}(\lambda)$ where the generator is G, and λ is the security parameter. Define the following attack game for a discrete log adversary $\mathcal{A}$:

- The challenger and the adversary $\mathcal{A}$ take in a description of $\mathbb{G}$, which includes the group order p and the generator G.
- The challenger chooses $s \xleftarrow{\$} \mathbb{Z}_p$, and gives $\mathcal{A}$ the group element $s \cdot G$.
- $\mathcal{A}$ outputs $\bar{s}$.

We say that $\mathcal{A}$ wins the game if $\bar{s} = s$ and we denote the probability of $\mathcal{A}$ winning the game as $\mathcal{E}_{\mathsf{DL}}[\mathcal{A}, \mathbb{G}]$. The discrete logarithm assumption holds if $\mathcal{E}_{\mathsf{DL}}[\mathcal{A}, \mathbb{G}]$ negligible in λ, i.e., $\mathcal{E}_{\mathsf{DL}}[\mathcal{A}, \mathbb{G}] \leq 1/P(\lambda)$ for any polynomial $P(\cdot)$.

Definition 1 (Security of signature scheme)**.** For a signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, define a following game for an adversary $\mathcal{A}$:

- The challenger runs $\mathsf{Gen}(1^\lambda) \to (SK, PK)$ and sends PK to $\mathcal{A}$.
- $\mathcal{A}$ asks the challenger for signatures on $M_1, \ldots, M_q$; and the challenger computes $\mu_i \leftarrow \mathsf{Sign}(SK, M_i)$ for $i = 1, \ldots, q$, and sends the corresponding signatures $\mu_1, \ldots, \mu_q$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a pair (M, μ).

We say that $\mathcal{A}$ wins the game if $M \notin \{M_1, \ldots, M_q\}$ and yet $\mathsf{Verify}(PK, M, \mu) = 1$. The advantage of $\mathcal{A}$ is the probability that $\mathcal{A}$ wins the game, denoted $\mathcal{E}_{\mathsf{forge}}[\mathcal{A}, \Sigma]$. We say that the signature scheme Σ is secure if, for any polynomial-time $\mathcal{A}$, $\mathcal{E}_{\mathsf{forge}}[\mathcal{A}, \Sigma]$ is negligible in λ.

G.1 Centralized Schnorr

Definition 2 (Schnorr signature scheme)**.** Let Hash be a hash function $\{0,1\}^* \to \mathbb{Z}_p$. Let $\mathbb{G}$ be a group of order p in which discrete log is hard. The Schnorr signature scheme consists of the following algorithms:

- $\mathsf{Gen}(1^\lambda) \to (SK, PK)$: a randomized algorithm run by the signer that takes in a security parameter λ, outputs $SK = s \xleftarrow{\$} \mathbb{Z}_p$ and $PK = S = s \cdot G$.
- $\mathsf{Sign}(s, M) :\to \mu$: a randomized algorithm run by the signer that on input a message M, a secret key $SK = s$, samples $r \xleftarrow{\$} \mathbb{Z}_p$, computes $R = r \cdot G$, $e = \mathsf{Hash}(S, R, M)$, and $\phi = r + es$. Output a signature $\mu := (R, \phi)$.
- $\mathsf{Verify}(S, M, \mu) \to v$: a deterministic algorithm run by the verifier that on input a public key $PK = S$, a message M, and a signature $\mu = (R, \phi)$, computes $e = \mathsf{Hash}(S, R, M)$ and checks if $\phi \cdot G = R + e \cdot PK$. If so, it outputs $v = 1$ (indicating the signature is valid). Otherwise, it outputs $v = 0$ (invalid signature).

Below we briefly recall the security proof of Schnorr signature scheme, because it helps in understanding the security proof of the threshold signing.

Theorem 3 ([37])**.** The Schnorr signature scheme is secure in the random-oracle model under the discrete logarithm assumption.

Proof. Assume that there exists an attacker $\mathcal{A}$ that can forge the signature, then we can build an attacker Sim that solves discrete log. The task of the attacker Sim is to simulate the view of $\mathcal{A}$ as in the real-world protocol execution (so that $\mathcal{A}$ can output a forgery), and then Sim transforms $\mathcal{A}$'s forgery into the ability to compute discrete log. We provide details below.

The discrete log challenger first samples a random s and sends $S = s \cdot G$ to Sim. The task for Sim is to utilize $\mathcal{A}$ to find s. In the first step of the simulation, $\mathcal{A}$ sends q messages $M_1, \ldots, M_q$ to Sim, and Sim needs to create signatures on these messages to send to $\mathcal{A}$. However, Sim cannot create such signatures "without any help" because it does not have the secret key s. To this end, we assume the hash is a programmable random oracle and Sim programs the random oracle Hash as follows: for each query (S, R, M) to Hash, it samples e, ϕ from $\mathbb{Z}_p$ at random, and then computes $R := \phi \cdot G - e \cdot S$ and set $\mathsf{Hash}(S, R, M)$ to e; and Sim will give $\mathcal{A}$ the tuple (R, ϕ) as the signature to m. When $\mathcal{A}$ verifies the signature, it computes $\phi \cdot G - R$, which equals $e \cdot S$ as random oracle Hash was programmed this way.

Now we show how $\mathcal{A}$ can create a forgery (R^*, ϕ^*) on a message M^*, where $M^* \notin \{M_1, \ldots, M_q\}$. In the random oracle model, we assume that $\mathcal{A}$ must have queried Hash on (S, R^*, M^*). Then Sim rewinds $\mathcal{A}$, and answers all the oracle queries before (S, R^*, M^*) with the same value used in the first run, but on the oracle query (S, R^*, M^*), Sim answers with a new e', randomly chosen from $\mathbb{Z}_p$ (note that R^* is the same as in the first run). In the second run, $\mathcal{A}$ outputs a forgery and if it happens to be M^* again, then Sim can compute the secret key s. Here Sim during rewinding can guess which message $\mathcal{A}$ will forge with uniform probability (choose where to rewind) so that the probability of success will be at least $1/\mathcal{Q}$, where $\mathcal{Q}$ is the total number of oracle queries made by $\mathcal{A}$. □

G.2 Threshold Schnorr for a single message

Recall the distributed key generation protocol (called JF-DKG as in GJKR [17]):

1. Each party i (acts as a dealer) chooses a random degree-t polynomial $\mathbf{H}_i(\cdot)$, where $r_i = \mathbf{H}_i(0)$ is the random value that party i wants to (additively) contribute, and $R_i = \mathbf{H}_i(0) \cdot G$ is the corresponding public value. Each party broadcasts 1) $\mathsf{ENC}_{PK_j}(\mathbf{H}_i(j))$, i.e., the encryption of share for party j under party j's public key; and 2) the Feldman commitments $\hat{\mathbf{H}}_i$ to $\mathbf{H}_i$.

   (Below we elide the distinction between committing to coefficients of $\mathbf{H}$ or to its values at sufficiently many points, since these are equivalent from a security perspective. We just use the fact that given $\hat{\mathbf{H}}_i$, it is possible to compute $\mathbf{H}_i(z) \cdot G$ for every $z \in \mathbb{Z}_p$. In particular, R_i is publicly known.)

2. The parties engage in an agreement protocol to agree on sets HOLD and QUAL such that $|\mathsf{HOLD}|, |\mathsf{QUAL}| \geq t+1$ and every party in HOLD received from every party in QUAL shares that are consistent with the committed polynomials.

3. Let $\mathbf{H} = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i$, each party $j \in \mathsf{HOLD}$ compute their share as $\rho_j = \mathbf{H}(j) = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i(j)$. The parties also compute a Feldman commitment to $\mathbf{H}$, $\hat{\mathbf{H}} = \sum_{i \in \mathsf{QUAL}} \hat{\mathbf{H}}_i$. The secret that is shared among them is $r = \mathbf{H}(0) = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i(0) \in \mathbb{Z}_p$. The corresponding public value is $R = \mathbf{H}(0) \cdot G = \sum_{i \in \mathsf{QUAL}} R_i \in \mathbb{G}$, which can be computed from $\hat{\mathbf{H}}$.

We use JF-DKG as a main component in our distributed Schnorr signature protocol, specifically, it is run to generate the ephemeral randomness that is needed for these signatures. Our protocol, described below, is similar (but not exactly the same as) the protocol from [17].

Protocol inputs. A message M to be signed, a degree-t sharing of a secret key s. Each party i holds a Shamir share of s, denoted as $\sigma_i = \mathbf{F}(i)$, with $\mathbf{F}$ a degree-t polynomial and $\mathbf{F}(0) = s$. Also, $\hat{\mathbf{F}}$ if publicly known, from which it is possible to compute $S = s \cdot G$ and $S_i = \sigma_i \cdot G$ for all i

Protocol outputs. A Schnorr signature of the form (R, ϕ) on the message M.

Protocol steps. The threshold signing mainly consists of three parts: first, generate ephemeral randomness for signing using JF-DKG; second, every party locally computes the hash and its Shamir share of the signature; finally, the parties combine the shares to reconstruct the signature.

1. The parties run the JF-DKG protocol above. After this, a set HOLD of parties is determined, where each party $j \in \mathsf{HOLD}$ holds a degree-t Shamir share $\rho_j = \mathbf{H}(j)$ of the ephemeral randomness $r = \mathbf{H}(0)$, where $\mathbf{H}$ is a degree-t polynomial, and everyone knows a Feldman commitment $\hat{\mathbf{H}}$ to $\mathbf{H}$.

   Let $R = \mathbf{H}(0) \cdot G$ and $\tilde{R}_j = \mathbf{H}(j) \cdot G$ for all $j \in \mathsf{HOLD}$. These can all be computed from the Feldman commitment $\hat{\mathbf{H}}$.

2. The parties locally compute $e = \mathsf{Hash}(S, R, M)$.

3. Each party $j \in \mathsf{HOLD}$ broadcasts its share of signature, $\pi_j = \mathbf{H}(j) + e \cdot \sigma_j$. Each share is verified by checking if $\pi_j \cdot G = \tilde{R}_j + e \cdot S_j$.

4. If at least $t+1$ parties in HOLD broadcast valid shares, use these shares to reconstruct $\phi = r + e \cdot s$. Output a signature on M as (R, ϕ).

The difference between this protocol and the one from [17] is that in our protocol, the parties in HOLD (that hold Shamir sharing of r, s) generate the signature, whereas in [17] it is the parties in QUAL that do it (using additive sharing of r, s). However, the same security challenge resides here as in our protocol, the adversary can play with QUAL for the ephemeral randomness (e.g., kick out a contributed polynomial of an honest party). The security proof is similar to that in [17], but not identical since the protocols are somewhat different.

G.2.1 Proof of Security

Definition 3 (Security of threshold signature)**.** Define the following game for a threshold signature protocol Π and an adversary $\mathcal{A}$ against a distributed signature protocol Π:

- $\mathcal{A}$ while interacting with Π, comes up with $M_1, \ldots, M_q$, and gets the corresponding q signatures $(\mu_1, \ldots, \mu_q)$.
- After interacting with Π, $\mathcal{A}$ outputs a pair (M, μ).

We say that $\mathcal{A}$ wins the game if $M \notin \{M_1, \ldots, M_q\}$ and yet $\mathsf{Verify}(PK, M, \mu) = 1$. The advantage of $\mathcal{A}$ is the probability that $\mathcal{A}$ wins the game, denoted as $\mathcal{E}_{\mathsf{forge}}[\mathcal{A}, \Pi]$. We say that a threshold signature protocol Π is secure if for any polynomial-time $\mathcal{A}$, $\mathcal{E}_{\mathsf{forge}}[\mathcal{A}, \Pi]$ is negligible.

Theorem 4. The threshold signature protocol from above is secure against a static adversary corrupting up to t parties, in the random-oracle model under the discrete logarithm assumption.

Proof. We describe a reduction, in which an adversary $\mathcal{A}$ against the threshold scheme can be used to solve the discrete logarithm problem. The main difference from the proof for centralized Schnorr is that the simulator must also simulate the adversary view of the protocol, not just of the signatures themselves. The main challenge in this proof is that the adversary can be rushing, which enables it to bias the distribution of the ephemeral randomness, so the simulator cannot just select the R at random.

Let Honest be the set of honest parties, let $\mathsf{Corrupt} = [n] \setminus \mathsf{Honest}$ be the corrupted parties, and let q be a bound on the number of random-oracle queries of the form (S, R, M) that $\mathcal{A}$ makes. The simulator needs to simulate for $\mathcal{A}$ the following aspects:

- The Feldman commitment $\hat{\mathbf{F}}$ to the degree-t polynomial $\mathbf{F}$ used to share the long-term key;
- The shares $\sigma_i = \mathbf{H}(i)$ of the long-term secret key for all $i \in \mathsf{Corrupt}$;
- The (encryption of) shares of ephemeral randomness that the honest parties send to the corrupted parties;
- The Feldman commitments $\hat{\mathbf{H}}_i$ to the honest parties' ephemeral randomness polynomials $\mathbf{H}_i$;
- The view of the agreement protocol, including the verifiable complaints and support messages;
- The full degree-t signature polynomial $\mathbf{Y}$ such that $\phi = \mathbf{Y}(0)$ is part of the signature (and $\mathbf{Y}$ is consistent with $\hat{\mathbf{F}}$ and the $\hat{\mathbf{H}}_i$'s);
- In addition to all the above, the simulator also needs to answer random-oracle queries (S, R, M) that $\mathcal{A}$ makes.

The reduction. The discrete log challenger randomly samples s from $\mathbb{Z}_p$ as the secret key and gives Sim the corresponding public key $PK = S = s \cdot G \in \mathbb{G}$. The public key is given to $\mathcal{A}$. Now Sim's task is to find s, by utilizing $\mathcal{A}$.

The simulator begins by choosing t random and independent scalars for the secret-key shares σ_i for all $i \in \mathsf{Corrupt}$ and giving them to $\mathcal{A}$. It also "interpolates in the exponent" a commitment to a degree-t polynomial which is consistent with the public key S and $\sigma_i \cdot G$ for all $i \in \mathsf{Corrupt}$.

In more detail, for each $z \in \mathsf{Corrupt} \cup \{0\}$ let $\mathbf{I}_z$ be the degree-t polynomial satisfying $\mathbf{I}_z(z) = 1$ and $\mathbf{I}_z(y) = 0$ for all $y \in \mathsf{Corrupt} \cup \{0\}$, $y \neq z$. Then the polynomial $\mathbf{F}$ is $\mathbf{F} = s \cdot \mathbf{I}_0 + \sum_{z \in \mathsf{Corrupt}} \sigma_z \cdot \mathbf{I}_z$. Denoting $\mathbf{F}_{\mathsf{Corrupt}} = \sum_{z \in \mathsf{Corrupt}} \sigma_z \cdot \mathbf{I}_z$, we can write $\mathbf{F} = \mathbf{F}_{\mathsf{Corrupt}} + s \cdot \mathbf{I}_0$, where the simulator knows $\mathbf{F}_{\mathsf{Corrupt}}$ in the clear.

Next, the simulator chooses at random an honest party $i^* \in \mathsf{Honest}$, which will be simulated differently than the other honest parties. Sim also chooses a random-oracle query index $\ell \in [q]$, hoping that the random-oracle query where $\mathcal{A}$ asks about (S, R, M) that are used in the signature is the ℓ'th query.

Throughout the simulation, the simulator answers random-oracle queries with independent random scalars $e_1, \ldots, e_q$, but e_ℓ will play a special role. Specifically, Sim chooses e_ℓ at the outset, together with another scalar $\tilde{\phi}$, and sets $R_{i^*} = \tilde{\phi} \cdot G - e_\ell \cdot S$. For the rest of the honest parties ($i \in \mathsf{Honest}$, $i \neq i^*$), Sim chooses r_i's at random and sets $R_i = r_i \cdot G$.

To simulate the Feldman commitments in Step 1 in JF-DKG, Sim simply follows DKG the protocol as prescribed for honest parties other than i^*. For party i^*, Sim needs to generate the commitment $\hat{\mathbf{H}}_{i^*}$ and the shares $\rho_{i^*j} = H_{i^*}(j)$ for $j \in \mathsf{Corrupt}$, without knowing the discrete lorarithm of R_{i^*}. These have to remain consistent, namely for all $j \in \mathsf{Corrupt}$ we need

$$\rho_{i^*j} \cdot G = R_{i^*} + \sum_{k=1}^{d} j^k \cdot (h_{i^*,j} \cdot G).$$

While Sim does not know the polynomial $\mathbf{H}_{i^*}$ in the clear (since its free term was chosen based on the unknown secret key), it can still create commitment $\hat{\mathbf{H}}_{i^*}$ that satisfies the relation above. Sim chooses at random $\rho_{i^*j} \leftarrow \mathbb{Z}_p$ for all $j \in \mathsf{Corrupt}$, and uses them as the shares of all the corrupted parties. Note that $\{\rho_{i^*j} \cdot G : j \in \mathsf{Corrupt}\}$, together with R_{i^*}, uniquely define the polynomial $\mathbf{H}_{i^*}$, which satisfies $\mathbf{H}_{i^*}(0) = \tilde{\phi} - e_\ell \cdot s$ and $\mathbf{H}_{i^*}(j) = \rho_{i^*j}$ for all $j \in \mathsf{Corrupt}$. Moreover, Sim can generate the commitment $\hat{\mathbf{H}}_{i^*}$ by "interpolating in the exponent", without needing to know $\mathbf{H}_{i^*}$ in the clear. Using similar notations to the above we can write

$$\mathbf{H}_{i^*} = \sum_{j \in \mathsf{Corrupt}} \rho_{i^*j} \cdot \mathbf{I}_j + (\tilde{\phi} - e_\ell \cdot s) \cdot \mathbf{I}_0 = \overbrace{\sum_{j \in \mathsf{Corrupt}} \rho_{i^*j} \cdot \mathbf{I}_j + \tilde{\phi} \cdot \mathbf{I}_0}^{=\mathbf{H}_{\mathsf{Corrupt}}} - e_\ell \cdot s \cdot \mathbf{I}_0, \tag{2}$$

where the simulator knows $\mathbf{H}_{\mathsf{Corrupt}}$ in the clear.

This completes the simulation of Step 1 of the DKG, the simulator sends all the shares and commitments of honest parties to the adversary. Then $\mathcal{A}$ sends back to Sim the polynomial $\mathbf{H}_i$ for parties $i \in \mathsf{Corrupt}$. (Sim gets the shares ρ_{ij} for $j \in \mathsf{Honest}$, and since it controls $t+1$ or more parties it can recover $\mathbf{H}_i$ in full).

Next, the simulator runs the prescribed agreement protocol on behalf of the honest parties, but treats i^* as an honest dealer, even though the ciphertexts that it broadcasts to the honest parties encrypt garbage. At the end of the agreement protocol, QUAL and HOLD are detemined, which in

turn defines also $\mathbf{H} = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i$ and $R = \mathbf{H}(0) \cdot G = \sum_{i \in \mathsf{QUAL}} R_i$. While the simulator does not know $\mathbf{H}$ in the clear, it does know the commitment to it $\hat{\mathbf{H}}$.

Sim aborts if either $i^* \notin \mathsf{QUAL}$,[25] or if $\mathcal{A}$ already made the random-oracle query (S, R, M) and it was not the ℓ'th query. (If $\mathcal{A}$ still did not make the query (S, R, M) by the time that R is defined, then the simulator will answer that query when it arrives with e_ℓ, regardless of the index of that query.)

Since there is at least one honest party in QUAL, then the first abort event happens with probability at most $(n-1)/n$. The second abort event happens with probability at most $(q-1)/q$, since there are at most q random-oracle queries. Hence, the simulator will proceed with probability at least $1/qn$. If the simulator did not abort, then we have

$$\mathbf{H} = \sum_{i \in \mathsf{QUAL} \setminus \{i^*\}} \mathbf{H}_i + \mathbf{H}_{i^*} = \overbrace{\sum_{i \in \mathsf{QUAL} \setminus \{i^*\}} \mathbf{H}_i + \mathbf{H}_{\mathsf{Corrupt}}}^{=\mathbf{H}_{\mathsf{CLR}}} - e_\ell \cdot s \cdot \mathbf{I}_0,$$

where the simulator knows $\mathbf{H}_{\mathsf{CLR}}$ in the clear.

Now the simulator proceeds to Step 4 of the signature protocol, where the honest parties broadcast their signature shares. Note that if the simulator did not abort, then we are ensured that $\mathsf{Hash}(S, R, M) = e_\ell$. The required signature is therefore $(R, r + e_\ell \cdot s)$, where $r = \mathbf{H}(0)$ is the discrete logarithm of R and $s = \mathbf{F}(0)$ is the discrete logarithm of S.

But the simulator needs to produce more than just the signature, it needs to come up with the full degree-t polynomial $\mathbf{Y} = \mathbf{H} + e_\ell \cdot \mathbf{F}$ (where the commitments $\hat{\mathbf{F}}, \hat{\mathbf{H}}$ to $\mathbf{F}$ and $\mathbf{H}$ are already fixed). Luckily, this is easy to do: Recall that $\mathbf{F} = \mathbf{F}_{\mathsf{Corrupt}} + s \cdot \mathbf{I}_0$ and $\mathbf{H} = \mathbf{H}_{\mathsf{CLR}} - e_\ell \cdot s \cdot \mathbf{I}_0$, and the simulator knows $\mathbf{F}_{\mathsf{Corrupt}}$, $\mathbf{H}_{\mathsf{CLR}}$ in the clear. Hence,

$$\begin{aligned} \mathbf{Y} &= \mathbf{H} + e_\ell \cdot \mathbf{F} = \mathbf{H}_{\mathsf{CLR}} - e_\ell \cdot s \cdot \mathbf{I}_0 + e_\ell \cdot (\mathbf{F}_{\mathsf{Corrupt}} + s \cdot \mathbf{I}_0) \\ &= \mathbf{H}_{\mathsf{CLR}} + e_\ell \cdot \mathbf{F}_{\mathsf{Corrupt}} \end{aligned}$$

which the simulator can output in the clear.

This concludes the simulation portion, and all that is left is to apply the forking lemma exactly as in the proof of the centralized Schnorr signature. Suppose $\mathcal{A}$ creates a forgery on M^* which is (R^*, ϕ^*). In the random oracle model, we assume that in order for $\mathcal{A}$ to generate such forgery it must have queried the oracle on (S, R^*, M^*). Let Sim rewind $\mathcal{A}$, changing the answer to the query (S, R^*, M^*). If $\mathcal{A}$ is still able to forge a signature on M^* with randomness R^*, then the simulator can extract s from those two signatures. □

G.3 Threshold Schnorr for multiple messages

Suppose we wanted to use the protocol from Appendix G.2 to sign multiple messages in parallel. A natural way of doing this would be to let each dealer D_i generate multiple polynomials $H_{u,i}$, $u \in [b]$, in b copies of the single-message protocol. However, the security of the parallel threshold Schnorr cannot be directly derived from the simulation proof for single-message protocol. Recall that in the proof in Appendix G.2.1, in order to generate valid signatures on the b messages $M^1, \ldots, M^b$, the simulator needs to guess the R's for the b messages, which brings down the succeeding probability

[25] While no shareholder will broadcast a complaint against party i^*, we cannot ensure that it is in QUAL since this is an asynchronous network and the adversary can delay the messages from party i^* until after QUAL is determined.

(for guessing the correct oracle queries) from $1/q$ to $1/q^b$. Below we only give a brief overview of how the proof changes; more details are found in Appendix G.6.

Mitigation and proof technique. To mitigate the security downgrade, we add a shift value δ to the ephemeral randomness where δ is determined after all the R's for the b messages are published. Specifically, let R_i^u be the randomness contributed by party $i \in [n]$ in the u-th copy where $u \in [b]$. After QUAL is determined[26], the randomness for the u-th messages is $R^u := \sum_{i \in \mathsf{QUAL}} R_i^u$. Then each party computes locally

$$\delta = \mathsf{Hash}\big(S, \{(R^u, M^u) : u \in [b]\}\big) \text{ and } \Delta = \delta \cdot G,$$

and the parties use $R^u + \Delta$ and $\phi^u + \delta$ for the signatures.

The intuition here is that $\mathcal{A}$ has very low probability of making a random-oracle query (S, R', M) with $R' = R^u + \Delta$ before δ is computed. The simulator, instead of guessing the queries of the form (S, R, M) for e, now guesses queries for δ of the form $(S, \{(R^u, M^u) : u \in [b]\})$.

The simulator aborts if the guess was wrong, or if $\mathcal{A}$ made a query $(S, R^{u\prime}, M^u)$ with the correct $R^{u\prime} = R^u + \delta \cdot G$ before step 2. (For each $u \in [b]$ the last event happens with probability at most $1/q$, and the total probability for the b messages can be upper bounded by the union bound.) If it did not abort, then the simulator knows all the $R^{u\prime}$'s, so it is free to program the random-oracle answers to all these queries $(S, R^{u\prime}, M^u)$.

G.4 Threshold Schnorr with a Super-Invertible Matrix

When using the super-invertible optimization, we still have each dealer D_i sharing a single polynomial $\mathbf{H}_i$, but we can derive b ephemeral randomness polynomials since we have at least b honest parties in QUAL.

However, having b honest parties in QUAL also means that we cannot use the same simulation strategy as above: Recall that in Appendix G.2.1, the simulator picks an honest party i^* at random, hoping that it will end up in QUAL. Sim simulates the actions of i^* differently from all the other honest parties, embedding a component that depends on S in the randomness R_{i^*}. Trying to do the same here, the simulator would have to guess not one but b honest parties from QUAL. Since in the asynchronous setting, we cannot ensure that honest parties end up in QUAL, the probability of guessing correctly is $1/\binom{n}{b}$.

Even worse, the simulator cannot know the linear combination to use for various quantities that the adversary expects to see until QUAL is determined: without the super-invertible optimization, the linear combination of the contributions from parties in QUAL was always a sum. But with this optimization, the linear combination is determined by a sub-matrix of the super-invertible Ψ, where the column corresponding to each party depends on QUAL.

To overcome these issues, we let the simulator embed S in the randomness $\mathbf{H}_i$ of *all the honest parties*, so it no longer needs to guess which honest parties will end up in QUAL. Moreover, we modify the protocol to include QUAL in the hash query for δ. This way, once the simulator sees the random-oracle query, it knows QUAL and can determine which party will correspond to what column of Ψ.

[26] Even though multiple polynomials are shared, we assume that the agreement protocol (Fig. 1, Step 2) guarantees the same QUAL for all of them.

G.5 Threshold Schnorr with Packing

The main difference induced by the packed variant is that the degrees of $\mathbf{F}$ and $\mathbf{H}$ are no longer the same. When describing the simulator, and in particular the way it sets up the randomness polynomials $\mathbf{H}_i$ of the honest parties, we can no longer just describe the relevant randomness scalars r and deduce the unique polynomial which is consistent with them. Instead, we construct the polynomials $\mathbf{H}_i$ in a form that would let the simulator cancel out the terms that depend on the secret key (that it doesn't know), and deduce the r's from them.

G.6 Putting it All Together

Combining all the modifications above, we next describe the threshold signature protocol with the super-invertible matrix optimization and packing.

Protocol inputs. $M^{1,1}, \ldots, M^{b,a}$ messages to be signed. Each party i holds a Shamir share of s, denoted as $\sigma_i = \mathbf{F}(i)$ where F has degree-$d = t + a - 1$ polynomial and $\mathbf{F}(0) = \mathbf{F}(-1) = \cdots = \mathbf{F}(1-a) = s$. Also, $S = s \cdot G$, $S_i = \sigma_i \cdot G$ as well as the Feldman commitment to $\mathbf{F}$ are public.

Protocol outputs. Schnorr signatures $(R^{u,v}, \phi^{u,v})$ for messages $M^{u,v}$ for all $u \in [b], v \in [a]$.

Protocol steps.

1. The parties run step 1 and step 2 of JF-DKG protocol, sharing polynomials $\mathbf{H}_i$ of degree $d' = t + 2a - 2$, and the agreement in step 2 ensures $|\mathsf{QUAL}| + |\mathsf{BAD}| = t + b$ and $\mathsf{HOLD} = t + d' + 1 = 2t + 2a - 1$. The polynomials $\{\mathbf{H}_i : i \in \mathsf{QUAL}\}$ will be used for generating b ephemeral-randomness polynomials, and we denote $r_{i,v} := \mathbf{H}_i(1-v)$ and $R_{i,v} := r_{i,v} \cdot G$.

2. Let $\Psi = [\psi_i^u]_{i \in \mathsf{QUAL}}^{u \in [b]} \in \mathbb{Z}_p^{b \times (t+b)}$ be super-invertible.

   For all $u \in [b]$, let $\mathbf{H}^u = \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i$.

   Each party $j \in \mathsf{HOLD}$ locally computes $\rho_j^u = \mathbf{H}^u(j) = \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i(j)$.

   From the Feldman commitments $\hat{\mathbf{H}}_i$ to $\mathbf{H}_i$ for $i \in \mathsf{QUAL}$, everyone can compute Feldman commitments $\hat{\mathbf{H}}^u$ to the $\mathbf{H}^u$'s. For those, anyone can compute the values $\mathbf{H}^u(j) \cdot G$ for all u, j, including $R^{u,v} = \mathbf{H}^u(1-v) \cdot G = \sum_{i \in \mathsf{QUAL}} \psi_i^u \cdot R_{i,v}$.

3. The parties locally compute $\delta := \mathsf{Hash}(S, \mathsf{QUAL}, \{(R^{u,v}, M^{u,v}) : u \in [b].v \in [a]\})$ and $\Delta = \delta \cdot G$. (Note the inclusion of QUAL in this hash query.)

4. For $u \in [b]$, run the packed signature protocol with randomness $\mathbf{H}^u$ and shift scalar δ: Everyone computes $e^{u,v} = \mathsf{Hash}(S, R^{u,v} + \Delta, M^{u,v})$ for all $v \in [a]$. Let $\mathbf{Z}^u$ be the unique degree-$a-1$ polynomials with $\mathbf{Z}^u(1-v) = e^{u,v}$ for all $v \in [a]$. Each party $j \in \mathsf{HOLD}$ sets $\pi_j^{u,v} = \mathbf{H}^u(j) + \mathbf{Z}^u(j) \cdot \mathbf{F}(j)$. Each signature share is verified as $\pi_j^{u,v} \cdot G = R_j^{u,v} + \mathbf{Z}(j) \cdot S_j$.

5. Reconstruct $\mathbf{Y}^u = \mathbf{H}^u + \mathbf{Z}^u \cdot \mathbf{F}$ from $\pi_j^u = \mathbf{Y}^u(j)$ for $j \in \mathsf{HOLD}$. For each $v \in [a]$ let $\phi^{u,v} = \mathbf{Y}(1-v)$, and output the signature

$$(R^{u,v} + \Delta, \phi^{u,v} + \delta).$$

Theorem 5. Under the discrete log assumption (Assumption 1), $\Pi_{\mathsf{super,\ pack}}$ (Figure 1) is a secure threshold signature protocol for generating ab signatures in the random oracle model, assuming a malicious adversary corrupting t parties.

Proof. The discrete log challenger randomly samples $s \leftarrow \mathbb{Z}_p$ as the signing secret key and gives the simulator the corresponding public key $S = s \cdot G$. The simulator's task is to find s, by utilizing $\mathcal{A}$. To use $\mathcal{A}$ the simulators needs to simulate for it the following aspects of the protocol:

- The Feldman commitment $\hat{\mathbf{F}}$ to the polynomial $\mathbf{F}$ of degree $d = t + a - 1$ that hides the long-term secret key;
- The shares $\sigma_i = \mathbf{F}(i)$ of the long-term secret key for $i \in \mathsf{Corrupt}$;
- The Feldman commitments $\hat{\mathbf{H}}_i$ to the polynomial $\mathbf{H}_i$ of degree $d' = t + 2a - 2$ from each party $i \in \mathsf{Honest}$;
- The (encryption of the) shares $\rho_{i,j} = \mathbf{H}_i(j)$ for every $i \in \mathsf{Honest}$ and $j \in \mathsf{Corrupt}$;
- The messages in the agreement protocol that decides $\mathsf{QUAL}, \mathsf{BAD}$ and HOLD, including the verifiable complaints and the support message;
- The b degree-d' polynomials $\mathbf{Y}_u$, $u \in [b]$, that must be consistent with the shares and with the Feldman commitments $\hat{\mathbf{F}}$ and the $\hat{\mathbf{H}}_i$'s;
- Sim also must answer all the oracle queries of $\mathcal{A}$, of the forms $(S, \mathsf{QUAL}, \{(R^{1,1}, M^{1,1}), \ldots, (R^{b,a}, M^{b,a})\})$ and (S, R, M).

The simulator needs to first simulate the shares of $\mathbf{F}$ and the Feldman commitments to $\mathbf{F}$, which is done similarly to Appendix G.2.1.

It chooses t random and independent scalars for the shares of the secret key, i.e., $\sigma_i = \mathbf{F}(i)$ for $i \in \mathsf{Corrupt}$, and gives them to $\mathcal{A}$. Now the simulator needs to create the Feldman commitments to F such that the share verification performed by $\mathcal{A}$ will pass. To do this, the simulator uses the public key $S = \mathbf{F}(0) \cdot G = \cdots = \mathbf{F}(1-a) \cdot G$, in conjunction with the t group elements $\sigma_i \cdot G$ for $i \in \mathsf{Corrupt}$: For each $z \in [1-a, 0] \cup \mathsf{Corrupt}$, denote by $\mathbf{I}_z$ the degree-d polynomial satisfying $\mathbf{I}_z(z) = 1$ and $\mathbf{I}_z(y) = 0$ for $y \in [1-a, 0] \cup \mathsf{Corrupt}$ and $y \neq z$. Denoting $\mathbf{I}^* = \mathbf{I}_0 + \mathbf{I}_{-1} + \ldots + \mathbf{I}_{1-a}$ and $F_{\mathsf{Corrupt}} = \sum_{z \in \mathsf{Corrupt}} \sigma_z \mathbf{I}_z$, the simulator defines the polynomial $\mathbf{F} = s \cdot \mathbf{I}^* + \mathbf{F}_{\mathsf{Corrupt}}$, where it knows $\mathbf{F}_{\mathsf{Corrupt}}$ in the clear.

Next, for each party $i \in \mathsf{Honest}$, the simulator picks two random polynomials, $\mathbf{A}_i$ of degree d' and $\mathbf{B}_i$ of degree $a - 1$, and computes the commitment to $\mathbf{H}_i$ as

$$\hat{\mathbf{H}}_i := \mathbf{A}_i \cdot G - \mathbf{B}_i \cdot \mathbf{I}^* \cdot S.$$

This corresponds to the polynomial $\mathbf{H}_i = \mathbf{A}_i + s \cdot \mathbf{B}_i \cdot \mathbf{I}^*$ (that the simulator does not know in the clear), and to public random elements

$$R_{i,v} = \mathbf{H}_i(1-v) \cdot G = \mathbf{A}_i(1-v) \cdot G - \mathbf{B}_i(1-v) \cdot \mathbf{I}^*(1-v) \cdot S$$

that everyone can compute from $\hat{\mathbf{H}}_i$.

Importantly, even though the simulator does not know $\mathbf{H}_i$ in the clear, it can compute $\mathbf{H}_i(j)$ in the clear for all $j \in \mathsf{Corrupt}$, since $\mathbf{I}^*(j) = 0$ and therefore $\mathbf{H}_i(j) = \mathbf{A}_i(j)$. This completes the simulation of step 1 in JF-DKG, and Sim sends to $\mathcal{A}$ the corresponding shares and the commitments.

Let q be the upper bound on the number of random-oracle queries for δ; the simulator chooses a random index $\ell \in [q]$, hoping that this will be the query where $\mathcal{A}$ asked on $(S, \mathsf{QUAL}, \{(R^{u,v}, M^{u,v}) : u \in [b], v \in [a]\})$ that are used for the signatures. All random oracle queries upto and including the ℓ'th query of that form are answered with fresh random scalars from $\mathbb{Z}_p$. Let QUAL^* be the value specified for QUAL in that ℓ'th query, and $R^{*,u,v}$ the groups elements in it. The simulator answers that query with a random $\delta \in \mathbb{Z}_p$, and denotes $\Delta = \delta \cdot G$. The simulator aborts if QUAL^* contains less than b honest parties, or if $\mathcal{A}$ made am earlier query $(S, R^{*,u,v} + \Delta, M^{u,v})$ for any $u \in [b], v \in [a]$. (The latter event can be upper bounded by the union bound, in Lemma 5.1 we show that it happens with probability at most abq/p. For the former event, we show bellow that it happens with probability at most $1 - 1/q$.)

The set QUAL^* implies the matrix $\Psi = [\psi_i^u]_{i \in \mathsf{QUAL}^*}^{u \in [b]}$. After Ψ and Δ and the $R^{*,u,v}$'s are defined, the Simulator defines the degree-$(a-1)$ polynomial

$$\mathbf{Z}^u = \sum_{i \in \mathsf{QUAL}^* \cap \mathsf{Honest}} \psi_i^u \mathbf{B}_i. \tag{3}$$

Random-oracle queries of the form $(S, R^{*,u,v} + \Delta, M^{u,v})$ are answered with $e^{u,v} = \mathbf{Z}^u(1 - v)$. Note that the polynomials $\mathbf{Z}^1, \ldots, \mathbf{Z}^b$ are just random and independent polynomials of degree $a-1$: The $\mathbf{B}_i$'s are random and independent, and also independent of the view of $\mathcal{A}$ (due to the $\mathbf{A}_i$'s that hide them). Moreover, there are at least b of them in $\mathsf{QUAL}^* \cap \mathsf{Honest}$, and the matrix Ψ is super-invertible. Hence, the $e^{u,v}$'s are random and independent, and so they are legitimate programming of the random oracle. All other queries are still answered with fresh random $\mathbb{Z}_p$ elements.

Next, the simulator runs the agreement protocol on behalf of honest parties; but treats them as honest dealers even if they broadcast garbage ciphertexts for each other. At the end of the agreement protocol, $\mathsf{QUAL}, \mathsf{BAD}$ and HOLD are determined such that $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq t + b$, $|\mathsf{BAD} \leq t|$, and $|\mathsf{HOLD}| \geq 2t + 2a - 1$. This in turn defines the ephemeral randomness $R^{u,v} = \sum_{i \in \mathsf{QUAL}} \psi_i^u R_{i,v}$ for each $u \in [b], v \in [a]$. If $\mathsf{QUAL} \neq \mathsf{QUAL}^*$ or $R^{u,v} \neq R^{*,u,v}$ for some u, v, then the simulator aborts. Since $\mathcal{A}$ makes at most q oracle queries (and we can assume w.l.o.g. that one of them is $(S, \mathsf{QUAL}, (R^{1,1}, M^{1,1}), \ldots, (R^{b,a}, M^{b,a})))$, then there is at least a $1/q$ chance that it does not abort. Note that since $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq t + b$ the QUAL includes at least b honest parties, hence $\mathsf{QUAL} = \mathsf{QUAL}^*$ implies that the first abort event from above did not happen either. Hence, Sim will proceed to the next steps with probability at least $\frac{1}{q}(1 - \frac{abq}{p}) = \frac{1}{q} - \frac{ab}{p}$.

If the simulator did not abort, we denote $\mathbf{H}^u = \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i$ for every $u \in [b]$. The simulator does not know the $\mathbf{H}^u$'s in the clear (since they depend on the secret key s). But since $\mathbf{H}_i = \mathbf{A}_i + s \cdot \mathbf{B}_i \cdot \mathbf{I}^*$ for $i \in \mathsf{Honest}$, we can write $\mathbf{H}^u$ as

$$\begin{aligned}
\mathbf{H}^u \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i &= \sum_{i \in \mathsf{QUAL} \cap \mathsf{Corrupt}} \psi_i^u \mathbf{H}_i + \sum_{i \in \mathsf{QUAL} \cap \mathsf{Honest}} \psi_i^u \mathbf{H}_i \\
&= \underbrace{\sum_{i \in \mathsf{QUAL} \cap \mathsf{Corrupt}} \psi_i^u \mathbf{H}_i + \sum_{i \in \mathsf{QUAL} \cap \mathsf{Honest}} \psi_i^u \mathbf{A}_i}_{= \mathbf{H}^u_{\mathsf{CLR}}} - s \left(\sum_{i \in \mathsf{QUAL} \cap \mathsf{Honest}} \psi_i^u \mathbf{B}_i \right) \cdot \mathbf{I}^*,
\end{aligned}$$

and Sim knows the full $\mathbf{H}^u_{\mathsf{CLR}}$ in the clear.

To simulate step 4 and 5 of $\Pi_{\mathsf{super,\ pack}}$, Sim needs to create in full the signature polynomials $\mathbf{Y}^1, \ldots, \mathbf{Y}^b$ of degree d'. For each $u \in [b]$, the u-th signature polynomial can be written as

$$\begin{aligned}
\mathbf{Y}^u &= \mathbf{H}^u + \mathbf{Z}^u \cdot \mathbf{F} \\
&= \mathbf{H}^u_{\mathsf{CLR}} - s \left(\sum_{i \in \mathsf{QUAL} \cap \mathsf{Honest}} \psi_i^u \mathbf{B}_i \right) \cdot \mathbf{I}^* \ + \ \mathbf{Z}^u \cdot (\mathbf{F}_{\mathsf{Corrupt}} + s \cdot \mathbf{I}^*) \\
&= \mathbf{H}^u_{\mathsf{CLR}} + \mathbf{Z}^u \cdot \mathbf{F}_{\mathsf{Corrupt}} \ + \ \underbrace{s \left(\mathbf{Z}^u - \sum_{i \in \mathsf{QUAL} \cap \mathsf{Honest}} \psi_i^u \mathbf{B}_i \right)}_{=0} \cdot \mathbf{I}^* \\
&= \mathbf{H}^u_{\mathsf{CLR}} + \mathbf{Z}^u \cdot \mathbf{F}_{\mathsf{Corrupt}},
\end{aligned}$$

which Sim known in the clear.

This completes the simulation, finally we apply the forking lemma as in Appendix G.2. □

Lemma 5.1 (Union bound for random oracle queries)**.** Suppose the random oracle outputs elements in a group $\mathbb{G}$ of order p. Let q be the number of queries of the form (S, R, M) that $\mathcal{A}$ made to the random oracle before δ is computed. Let ab be the number of messages to be signed, and denote by $R^{u,v}$, $u \in [b], v \in [a]$, the group elements that are included in the query where δ is computed. Then the probability that $R^{u,v} + \delta \cdot G$ is included in any $\mathcal{A}$'s queries before δ is computed is at most abq/p.

Proof. Let $\mathcal{R} := \{R \in \mathbb{F} : R \text{ was included in one of the } q \text{ queries}\}$. By definition, we know that $|\mathcal{R}| \leq q$. For any element $X \in \mathbb{G}$, let $\mathcal{R} - X = \{R - X : R \in \mathcal{R}\}$. Then we have $|\mathcal{R} - X| \leq q$ for every $X \in \mathbb{G}$, and therefore:

$$\begin{aligned}
&\Pr[\exists u \in [b], v \in [a] \text{ such that } R^{u,v} + \delta G \in \mathcal{R}] \\
&= \Pr[\exists u \in [b], v \in [a] \text{ such that } \delta G \in \mathcal{R} - R^{u,v}] \\
&\leq \sum_{u \in [b], v \in [a]} \Pr[\delta G \in \mathcal{R} - R^{u,v}] \ \leq \ ab \cdot (q/p).
\end{aligned}$$

□

G.7 Security for the Dynamic Setting

The main difference between the dynamic and the static cases is that in the dynamic case, it is the shareholders that need to generate the signatures at the end, while the dealers hold the shares of the long-term secret key at the beginning. The sets of dealers and shareholders are arbitrary, they can be the same set, disjoint sets, or anywhere in between. Hence, the dynamic setting requires a re-share protocol in order for the dealers to pass the shared secret key to the shareholders. (In the static case the dealers and shareholders are the same set, hence no re-sharing is needed.)

Note that even in the dynamic case we assume that the secret key is uniformly random and cannot be biased by the adversary. This can be implemented by having a trusted party share it in the first place, or using a non-biased DKG protocol (e.g., the protocol from [17] that uses statistically-hiding commitments). The Shamir sharing of that unbiased key is an input to the

protocol. While the adversary may bias the new shares if we use Feldman commitments, it cannot bias the key itself.

The simulation proof for the dynamic case is mostly the same as the static case, so we only describe briefly the added components. Specifically, the simulator needs to simulate also the degree-d re-sharing polynomials F_i's from the honest parties. Recall that the simulator in Appendix G.6 generates a Feldman commitment $\hat{\mathbf{F}}$ for $\mathbf{F}$, this allows it to compute $S_i\mathbf{F}(i) \cdot G$ for every i. Then it can write $\mathbf{F}_i$ just like it did $\mathbf{F}$, namely $\mathbf{F}_i = \mathbf{F}_{i,\mathsf{Corrupt}} + \mathbf{F}(i) \cdot \mathbf{I}^*$, where it knows $\mathbf{F}_{i,\mathsf{Corrupt}}$ in the clear. The rest of the proof follows, with the simulator setting the re-shared polynomial $\mathbf{F}'$ just like it did the $\mathbf{H}^u$'s (except using the Lagrange coefficients λ_i rather than the matrix entries ψ_i^u).

H Mixed Adversary Model and Dishonest Majority

Focusing on the honest super-majority case (i.e., $n > 3t$) in our analysis is necessary to preserve the security of SPRINT in the asynchronous setting, including the robustness of signatures and the safety of the global secret across refreshes. Here, we remark on the security of the SPRINT protocol from Figs. 1 and 2 in some cases where $n \leq 3t$ or even with dishonest majority $n < 2t$. Specifically, we consider the *mixed adversary* model of Hirt et al. [25] that distinguishes between malicious parties (that can deviate from the protocol in arbitrary ways) and honest-but-curious ones (that run the protocol as specified but whose internal secrets are available to the attacker). In this model, one can get more relaxed bounds on the number of dishonest parties, including security against a dishonest majority (involving both malicious and honest-but-curious parties), while preserving robustness.

Denote by h (a lower bound on) the number of honest parties, and m, c (upper bounds on) the number of malicious and honest-but-curious parties, respectively (here $t = m + c$). We focus on the static case, Fig. 1, as the bounds apply to the dynamic case too. Given these parameters (with $n = h + c + m$) and the packing parameter a, we run the protocol from Fig. 1 using polynomials of degrees $d = m + c + a - 1$ (for the long-term secret) and $d' = m + c + 2a - 2$ (for the ephemeral randomness), and with $d_1 = |\mathsf{QUAL}| = d_0 = |\mathsf{HOLD}| = n - m$ for the agreement-protocol parameters. To be able to sign, we must ensure that $|\mathsf{HOLD}| - m = d_0 - m \geq d' + 1$. Substituting $d' = m + c + 2a - 2$ and $d_0 = n - m$ we get $n - 2m \geq m + c + 2a - 1$, which means $a \leq (n - c - 3m + 1)/2$. For the parameter b, we need b honest parties in QUAL so $b = |\mathsf{QUAL}| - m - c = n - 2m - c$.

Consider the dishonest-majority example $n = 100, h = 49, m = 20, c = 31$, then we can set $a = 5, b = 29$ and we can produce 145 signatures in one run of the protocol with 49% of honest parties. In another example: $n = 16, h = 9, m = 3, c = 4$, we get $a = 2, b = 6$ so 12 signatures. This is just over one quarter of what can be achieved with $n = 16, m = 3$ but security here withstands 4 additional honest-but-curious participants.

I Refreshing Packed Secrets

When describing SPRINT, we focused on how to refresh packed secrets in which all the scalars are equal, i.e., a vector of secrets of the form $(s, \ldots, s)$. For our application to high-throughput Schnorr signatures that was enough, since we needed to put the same secret key in all these slots. But other applications may want to maintain shares of more general vectors, of the form $(s_1, s_2, \ldots, s_a)$, where the s_v's can be different from one another. For the sake of completeness, we describe here a simple method for refreshing sharing of these more general packed secrets.

Below, let I_u for $u = 1, 2, \ldots, a$ the unique polynomial of degree $a-1$ satisfying $I_u(1-u) = 1$ and $I_u(1-v) = 0$ for all $v \in \{1, \ldots, a\}, v \neq u$. Refreshing a packed sharing of $(s_1, s_2, \ldots, s_a)$ roughly consists of sharing a different polynomials, each containing just one of the s_v's, then using the I_u's to combine them into a single packed polynomial. We describe below two variants of this approach: One having each dealer share only a single polynomial, and results in a packed polynomial of the slightly larger degree of $t + 2a - 1$. The other has each dealer share a different polynomials which results in a packed polynomial of degree $t + a$ (which is the smallest possible).

I.1 Method One: Sharing One Polynomial

In this method, we maintain the invariant that the vector of secrets $(s_1, \ldots, s_a)$ is shared using a packed degree-$(t + 2a - 2)$ polynomial F with $F(1-v) = s_v$ for all $v = 1, 2, \ldots, a$. (Note that even though we offer robustness against t corrupted parties and only pack a values, the polynomial that we keep has degree $t + 2a - 2$ and not just $t + a - 1$.)

To refresh, each party re-shares its share via a packed polynomial of degree $t + a - 1$. Namely, F_i of degree $t + a - 1$ such that $F_i(0) = F_i(-1) = \cdots = F_i(1-a) = \sigma_i$. Each shareholder j gets the sub-share $\sigma_{ij} = F_i(j)$ from each dealer i.

The shareholders then proceed with the usual agreement protocol, agreeing on a set QUAL of qualified dealers with cardinality $|QUAL| = t + a$, and a set HOLD of shareholders with cardinality $|\mathsf{HOLD}| \geq 2t + a$, such that all the shareholders in HOLD have valid sub-shares from all the dealers in QUAL.

Next, a shareholder $j \in \mathsf{HOLD}$ first computes a different temporary shares corresponding to a polynomials that hold the a different values. Specifically, for $v = 1, 2, \ldots, a$ consider the Lagrange coefficients $\{\lambda_{iv} : i \in \mathsf{QUAL}, v = 1, \ldots, a\}$, such that for every degree-$(t+a-1)$ polynomial $P(X)$ it holds for all $v = 1, \ldots, a$ that $P(1-v) = \sum_{i \in QUAL} \lambda_{i,v} P(i)$. Party j computes for all $v = 1, 2, \ldots, a$

$$\rho_{j,v} = \sum_{i \in QUAL} \lambda_{i,v} \sigma_{ij}.$$

Let us denote $F'_v = \sum_{i \in QUAL} \lambda_{i,v} F_i$, then clearly all the F'_v's are degree-$(t + a - 1)$ polynomials with $\rho_{j,v} = F'_v(j)$, and

$$F'_v(1-v) = \sum_{i \in \mathsf{QUAL}} \lambda_{i,v} F_i(1-v) = \sum_{i \in \mathsf{QUAL}} \lambda_{i,v} F(i) = F(1-v) = s_v.$$

Party P_j computes its final share as $\sigma'_j = \sum_{v=1}^{a} \rho_{j,v} \cdot I_v(j)$. Clearly, this is indeed a share on the polynomial $F' = \sum_{v=1}^{a} I_v \cdot F'_v$ of degree $t + 2a - 2$, and for all $v = 1, 2, \ldots, a$ we have

$$F'(1-v) = \sum_{v'=1}^{a} I_{v'}(1-v) \cdot F'_{v'}(1-v) = F'_v(1-v) = s_v,$$

as needed.

We remark that by construction, the polynomial F' cannot reveal more that what's implied by the polynomials $F'_1, \ldots, F'_a$, which in turn only reveal the s_v's.

I.2 Method Two: Sharing a Polynomials

In this method, we maintain the invariant that the vector of secrets $(s_1, \ldots, s_a)$ is shared using a packed degree-$(t+a-1)$ polynomial F with $F(1-v) = s_v$ for all $v = 1, 2, \ldots, a$.

Each dealer D_i has a share $\sigma_i = F(i)$, and they prepare a degree-t polynomials $F_{i,1}, \ldots, F_{i,a}$, random subject to the condition that $F_{i,v}(1-v) = \sigma_i$ for all $v \in [a]$. D_i shares these polynomials, with shareholder P_j receiving $\sigma_{i,j,v} = F_{i,v}(j)$ for all $v \in [a]$.

The shareholders agree on sets HOLD, $\mathsf{QUAL}_1, \ldots, \mathsf{QUAL}_v$ such that for all $v \in [a]$, all the shareholders in HOLD have valid shares of $F_{i,v}$ from all the dealers in QUAL_v, and in addition $|\mathsf{HOLD}| \geq t + a$ and $|\mathsf{QUAL}_v| \geq t + 1$ for all $v \in [a]$.

For all $v \in [a]$, let $\{\lambda_{i,v} : i \in \mathsf{QUAL}_v\}$ be the Lagrange coefficients for recovering $F(1-v)$ from $\{F(i) : i \in \mathsf{QUAL}_v\}$. Each shareholder $P_j \in \mathsf{HOLD}$ computes a share $\sigma'_{j,v} = \sum_{i \in \mathsf{QUAL}_v} \lambda_{i,v} \sigma_{i,j,v}$. This share is $\sigma'_{j,v} = F'_v(j)$, where F'_v is the degree-t polynomial $F'_v = \sum_{i \in \mathsf{QUAL}_v} \lambda_{i,v} F_{i,v}$. As usual, it is easy to see that we have $F'_v(1-v) = F(1-v) = s_v$ for all $v \in [a]$.

Finally, each shareholder P_j sets $\sigma'_j = \sum_{v=1}^{a} I_v(j) \cdot \sigma'_{j,v}$. Clearly, we have $\sigma'_j = F'(j)$ for the polynomial $F' = \sum_{v=1}^{a} I_v \cdot F'_v$ of degree $t + a - 1$, and $F'(1-v) = s_v$ for all $v \in [a]$.

J The Parameter-Finding Utility

See https://shaih.github.io/pubs/committee_sizes.py.txt for the parameter finding utility. It computes the required committee sizes, when sub-sampling committees, e.g., in a blockchain environment. See Appendix D.

J.1 Example Parameters

```
$ python math/committee_sizes.py
Optimistic Setting:
number of parties:         n = 676
threshold for safety:      t = 250
packing parameter:         a = 64
corrupt fraction liveness: alpha = 5.0 %
corrupt fraction safety:   alpha = 20.0 %
liveness error:            4.35e-03 = 2^-7.85
safety error:              5.89e-25 = 2^-80.49
safety errors for higher alpha:
  for alpha = 21.43 %:     1.00e-20 = 2^-66.4
  for alpha = 22.86 %:     4.88e-17 = 2^-54.2
  for alpha = 24.29 %:     7.71e-14 = 2^-43.6
  for alpha = 25.72 %:     4.42e-11 = 2^-34.4
  for alpha = 27.16 %:     1.01e-08 = 2^-26.6
  for alpha = 28.59 %:     9.86e-07 = 2^-20.0
  for alpha = 30.02 %:     4.43e-05 = 2^-14.5
  for alpha = 31.45 %:     9.71e-04 = 2^-10.0
=======================================================

Pessimistic Setting:
number of parties:         n = 992
threshold for safety:      t = 336
packing parameter:         a = 40
corrupt fraction liveness: alpha = 20.0 %
corrupt fraction safety:   alpha = 20.0 %
```

```
liveness error:            4.12e-04 = 2^-11.24
safety error:              5.95e-25 = 2^-80.48
safety errors for higher alpha:
  for alpha = 21.17 %:     8.78e-21 = 2^-66.6
  for alpha = 22.35 %:     4.01e-17 = 2^-54.5
  for alpha = 23.52 %:     6.29e-14 = 2^-43.9
  for alpha = 24.69 %:     3.70e-11 = 2^-34.7
  for alpha = 25.86 %:     8.79e-09 = 2^-26.8
  for alpha = 27.03 %:     9.01e-07 = 2^-20.1
  for alpha = 28.21 %:     4.24e-05 = 2^-14.5
  for alpha = 29.38 %:     9.63e-04 = 2^-10.0
=========================================================
```